UNIVERSITÀ DEGLI STUDI ROMA TRE

Dipartimento di Ingegneria

Corso Di Laurea Triennale In

INGEGNERIA INFORMATICA

PROGETTAZIONE E SVILUPPO DI METODI EFFICIENTI PER LA REALIZZAZIONE DI COMUNICAZIONI NASCOSTE SU PIATTAFORME MOBILI ANDROID TRAMITE L'UTILIZZO DI CODICE NATIVO C E PROTOCOLLO CONNECTIONLESS

Tesi di Laurea di

Davide Di Maria

Relatore

Francesco Benedetto

Correlatori

Angelo Liguori

Antonio Tedeschi

Anno Accademico 2014/2015

Alla mia famiglia

Premessa

La sicurezza è uno degli obiettivi fondamentali che si cerca di perseguire in qualunque sistema informatico.
La diffusione di internet e di dispositivi come i personal computer o gli smartphone, infatti, ha spinto molte aziende e utenti a gestire i loro dati sensibili su di essi, trasformandoli, di fatto, in delle vere e proprie banche dati.
Lo studio di metodologie atte a difendere queste informazioni, da corruzione o dalla possibile fruizione di esse da utenti non autorizzati, risulta quindi necessario.
Nel corso della trattazione di questa tesi verranno descritte le tipologie di attacchi informatici che sono state sviluppate per perseguire questo scopo, riservando una particolare attenzione allo studio di un particolare tipo di attacco informatico denominato Covert Channel.
Questo tipo di attacco, che rappresenta l'argomento centrale di questa tesi, permette la trasmissione di informazioni illecite tra due parti comunicanti in maniera occultata ad un eventuale organo di controllo, fattore che li rende una minaccia elevata per la sicurezza di qualsiasi sistema.
Questo scopo viene perseguito dai Covert Channel tramite l'alterazione delle risorse "lecite" di un sistema informatico attraverso delle modalità diverse in base alla tipologia di canale nascosto utilizzato.
I Covert Channel vengono suddivisi in Covert Storage Channel e Covert Timing Channel.
La differenza tra queste due categorie risiede nella modalità operativa adottata per mascherare la comunicazione illecita.
I Covert Storage Channel basano il loro funzionamento sull'occultamento di informazioni in aree di memoria, di solito utilizzate per altri scopi, accessibili dal destinatario della comunicazione.
I Covert Timing Channel, invece, fondano il loro funzionamento sulla temporizzazione di particolari eventi, la cui avvenuta o meno in un determinato intervallo temporale comunica una particolare informazione ad un destinatario in grado di osservare la sequenzialità degli eventi.
Nel corso della trattazione di questa tesi verranno esaminati gli algoritmi principali utilizzati per la realizzazione di questi canali nascosti, ma anche le contromisure adottate per renderli meno efficaci.
Verrà, inoltre, presentata la progettazione e i passi di realizzazione di un Covert Timing Channel sulla piattaforma mobile Android, sistema operativo mobile scelto per la sua larghissima diffusione.
Nel corso della trattazione della tesi verranno descritte le metodologie utilizzate per l'implementazione di questo canale nascosto, esponendo le problematiche insorte durante lo sviluppo di questa applicazione.
Verranno inoltre riportati i dati riguardanti la sperimentazione di questa applicazione che ne metteranno in risalto le prestazioni in diverse condizioni.

Il presente lavoro di tesi si compone dei seguenti quattro capitoli.
Nel primo capitolo verranno introdotti i concetti della sicurezza informatica, esaminandone gli obiettivi e le nozioni fondamentali.

Una particolare attenzione verrà riservata al protocollo multilivello, una metodologia di difesa la cui affidabilità, elevata visto l'impiego di questo protocollo anche da enti governativi, è stata messa in discussione dai canali nascosti, in grado di violarne i principi, mettendone a rischio la funzionalità.

Il secondo capitolo è interamente dedicato ai Covert Channel di cui verrà esaminata in dettaglio la classificazione, descrivendo i punti di forza e le debolezze delle varie tipologie di canale nascosto.
L'attenzione del lettore sarà canalizzata sui Covert Timing Channel, di cui verranno presentate alcune note implementazioni, introducendo, successivamente, alcuni degli algoritmi che hanno fornito la base su cui si è poggiata l'implementazione del canale nascosto su Android.

Nel terzo capitolo, dedicato all'implementazione di un Covert Channel, verrà descritta la progettazione del sistema, illustrando gli algoritmi e le procedure che hanno permesso la realizzazione di un canale nascosto in ambito Android, con una disamina dettagliata delle problematiche incorse durante la fase di sviluppo.
Alcune di queste hanno riguardato l'inserimento di codice nativo C nell'applicazione Android e l'introduzione di metodologie atte al mantenimento di una giusta sincronizzazione tra le due parti comunicanti.

Nel quarto ed ultimo capitolo verranno, infine, analizzate le prestazioni del Covert Timing Channel realizzato nel presente progetto di tesi, osservando la variazione delle performance alla modifica degli input necessari per il funzionamento del canale nascosto.

Indice

1 Introduzione sulla sicurezza informatica 12

1.1. OBIETTIVI DELLA SICUREZZA INFORMATICA 12
1.2. CLASSIFICAZIONE DEGLI ATTACCHI INFORMATICI 14
1.2.1. Denial of Service (DoS/DDoS) 15
1.2.2. DNS Cache Poisoning 17
1.2.3. The Man in the Middle Attack 18
1.3. PRINCIPALI METODOLOGIE DI DIFESA 19
1.3.1. Il protocollo di sicurezza multilivello (MLS) 19
1.4. SICUREZZA NEI SISTEMI MOBILI 23
1.4.1. Attacchi tramite SMS/MMS 23
1.4.2. Attacchi tramite WI-FI e Bluetooth 24
1.4.3. Attacchi basati sul sistema operativo 24
1.4.4. Attacchi tramite malware 25
1.5. CONTROMISURE E STRATEGIE DI DIFESA 25
1.5.1. Sicurezza nei sistemi operativi mobili 25
1.5.2. Software per la protezione 26

2 I Covert Channel 28

2.1. L'ANALOGIA DEI DUE PRIGIONIERI 28
2.2. TIPOLOGIE DI COVERT CHANNEL 29
2.2.1. Covert Storage Channels 29
2.2.2. Covert Timing Channels 31

3 Implementazione di un Timing Covert Channel 36

3.1. SCOPI ED OBIETTIVI 36
3.1.1. Modello Client/Server 37
3.1.2. Tecnologie e strumenti utilizzati 39
3.2. IMPLEMENTAZIONE 40
3.2.1. Realizzazione in linguaggio C 40
3.2.2. Struttura dell'applicazione Android 46
3.3. SINCRONIZZAZIONE 51
3.3.1. Start of Frame delimiter 52

4 Analisi delle prestazioni 55

4.1. DISPOSITIVI UTILIZZATI 55
4.2. PRESTAZIONI IN LOCAL HOST 56
4.2.1. Prestazioni del Covert channel realizzato in Android 56
4.2.2. Prestazioni del Covert Channel realizzato in C 56

4.3. PRESTAZIONI DEL COVERT CHANNEL TRA DUE DISPOSITIVI FISSI 57
4.4. PRESTAZIONI DEL COVERT CHANNEL IMPLEMENTATO SU ANDROID 58

Riferimenti **69**

Indice delle Figure

Figura 1 Gli obiettivi principali perseguiti dalla sicurezza Informatica [2] 13
Figura 2 Classificazione dei Malware [5] 14
Figura 3 Il funzionamento di un attacco di tipo Dos [7] 16
Figura 4 Funzionamento del DNS Cache Poisoning [8] 17
Figura 5 Scenario classico del MITM attack [9] 18
Figura 6 I diversi livelli di sicurezza [12] 20
Figura 7 esempio di Modello Bell – LaPadula [12] 22
Figura 8 Classico funzionamento del modello Biba [14] 22
Figura 9 Informazioni salvate su un dispositivo mobile [15] 23
Figura 10 Il problema dei due prigionieri [22] 28
Figura 11 Classici campi di un pacchetto IP [24] 30
Figura 12a Funzionamento di un Remote Storage Covert Channel [26] 31
Figura 13 Trasmissione secondo l'ON/OFF scheme [27] 33
Figura 14 Capacità dei covert timing channel di passare inosservati a organi di difesa 34
Figura 15 Problemi di sincronizzazione nell' ON/OFF scheme [27] 34
Figura 16 Grafico che mostra l'evoluzione del mercato degli OS mobili [31] 36
Figura 17 Tipica comunicazione tra un client ed un server [33] 37
Figura 18 *Un esempio di Android.mk [35]* 40
Figura 19 Tipica comunicazione client e server in UDP [36] 41
Figura 20 La funzione recvfrom si pone in attesa fino all'arrivo di un datagramma [36] 45
Figura 21 Grafico che mostra la relazione tra time slot e prestazioni del covert channel 61
Figura 22 Il grafico mostra il rapporto tra le prestazioni e il time slot per la sequenza numerica 7155493201 64
Figura 23 rapporto tra prestazioni ed errori per la trasmissione del messaggio con gli spazi . 66
Figura 24 Prestazioni osservate nell'invio dei 3 messaggi 67

Indice delle Tabelle

Tabella 1 Market Share del mercato mobile negli ultimi anni [31] 37
Tabella 2 Paragone tra le funzionalità offerte dai due IDE [34] 39
Tabella 3 Librerie impiegate dal Client in C 41
Tabella 4 Il metodo che si occupa della codifica in binario di un carattere 42
Tabella 5 metodo che codifica una stringa in binario 42
Tabella 6 trasmissione implementata secondo l'on/off scheme 43
Tabella 7 Creazione di una socket per la comunicazione 44
Tabella 8 implementazione della decodifica degli intervalli temporali osservati dal server ... 45

Tabella 9 Metodo che si occupa della decodifica del messaggio covert 46
Tabella 10 permessi necessari per l'accesso a internet dell'applicazione Android 47
Tabella 11 Librerie utilizzate per lo stabilimento della connessione tramite socket 48
Tabella 12 metodo che stabilisce la comunicazione tra client e server 48
Tabella 13 metodo che lancia la SendActivity una volta ricevuta una risposta positiva dal server 49
Tabella 14 Listato dei metodi principali della MainActivity 49
Tabella 15 Metodo che permette il caricamento delle librerie in C nel codice Java 50
Tabella 16 Metodo JNI che fa da tramite tra linguaggio Java e C 50
Tabella 17 Corpo del metodo JNI 51
Tabella 18 Listato dei metodi della SendActivity 51
Tabella 19 Metodo per l'invio del pattern di sincronizzazione 52
Tabella 20 Metodo per la decodifica lato server 53
Tabella 21 Metodo che aggiusta il sincronismo dei pacchetti arrivati 53
Tabella 22 Invio del messaggio con un time slot di 7 ms 57
Tabella 23 Invio della stringa a 5 ms 57
Tabella 24 Trasmissione del messaggio “mario.rossi@gmail.com” con un intervallo di 100 ms 58
Tabella 25 Invio del messaggio “mario.rossi@gmail.com” con un timing slot di 75 ms 59
Tabella 26 Invio del messaggio "mario.rossi@gmail.com" a 50 ms 60
Tabella 27 Invio del messaggio "mario.rossi@gmail.com" a 25 ms 60
Tabella 28 Dati dell'invio del messaggio "mario.rossi@gmail.com" a 10 ms 61
Tabella 29 Invio del codice numerico a 100 ms 62
Tabella 30 Invio del codice numerico a 75 ms 62
Tabella 31 Invio del codice numerico a 50 ms 63
Tabella 32 Invio della sequenza numerica a 25 ms 63
Tabella 33 Invio del messaggio a 10 ms 63
Tabella 34 Invio del messaggio a 100 ms 64
Tabella 35 Trasmissione dell'informazione a 75 ms 65
Tabella 36 Trasmissione a 50 ms 65
Tabella 37 Trasmissione del messaggio a 25 ms 65
Tabella 38 Trasmissione del messaggio a 10 ms 66

1 Introduzione sulla sicurezza informatica

Il termine sicurezza informatica indica quel ramo dell'informatica che si occupa dell'analisi delle vulnerabilità, dei rischi e dei possibili attacchi atti a minare l'integrità fisica (hardware) e logico-funzionale (software) di un sistema informatico.
La necessità di una metodologia di protezione delle informazioni ha radici antiche e durante il corso degli anni è mutata di pari passo con l'evoluzione delle tecnologie impiegate.
Durante la Seconda Guerra Mondiale e i primi anni '50 gli obiettivi primari erano codificare i dati in modo tale che il nemico non potesse interpretarli, famosissima la decrittazione del codice Enigma nazista da parte di Turing, e proteggere le sedi fisiche degli stessi, limitando l'accesso al personale autorizzato.
Negli anni '70 e '80 con lo sviluppo di ARPANET, embrione della comunicazione internet, si cominciarono ad analizzare le prime debolezze di una rete di comunicazione come la mancanza di una autorizzazione e una identificazione degli utenti.
La diffusione capillare di computer e di internet nella fine degli anni '90 e nei primi anni del ventunesimo secolo, ha portato molte aziende ed enti governativi a gestire le loro informazioni attraverso l'utilizzo dei computer ed a trasmetterle tramite la rete, rendendo la problematica della sicurezza molto più impellente.
Il sistema informatico deve essere in grado, quindi, di impedire l'alterazione diretta o indiretta delle informazioni, sia da parte di utenti non autorizzati, sia a causa di eventi accidentali, impedendo, inoltre, l'accesso abusivo ai dati.

1.1. Obiettivi della sicurezza informatica

Gli obiettivi [1], illustrati in Figura 1, generalmente perseguiti dalla sicurezza informatica sono:

1. *Integrità,* ovvero nel determinare che i dati inseriti non siano stati manipolati da agenti esterni o durante la comunicazione;
2. *Confidenzialità,* ovvero rendere leggibili i dati solamente agli attori della transizione;
3. *Disponibilità,* il cui scopo è quello di garantire un accesso al sistema di informazione e di mantenerne il corretto funzionamento;
4. *Il non ripudio,* la garanzia che nessuno dei corrispondenti possa negare la transazione;
5. *Autenticazione,* che consiste nel certificare l'identità di un utente, assicurando ad ogni utilizzatore di conoscere l'identità della persona con cui sta comunicando.
6. *Privacy*, la garanzia che le informazioni private degli utenti restino tali.

Per raggiungere questi obiettivi viene svolta la cosiddetta *analisi dei rischi*, ovvero l'analisi delle possibili *minacce* e delle *vulnerabilità* di un sistema informativo. In questo contesto, il

termine *minaccia* indica un tipo di azione suscettibile a nuocere in assoluto mentre con *vulnerabilità* si denota il livello di esposizione alla minaccia in un determinato contesto.
Infine con il termine *contromisura* si indicano le azioni che devono essere effettuate per prevenire le minacce.
Le contromisure da attuare non sono unicamente delle soluzioni tecniche ma anche delle misure di formazione e di sensibilizzazione rivolte agli utenti, un insieme di regole ben definite atte alla prevenzione di un attacco informatico.
Il rischio viene di conseguenza definito come:

$$\text{Rischio} = \frac{Minaccia * Vulnerabilità}{Contromisure}$$

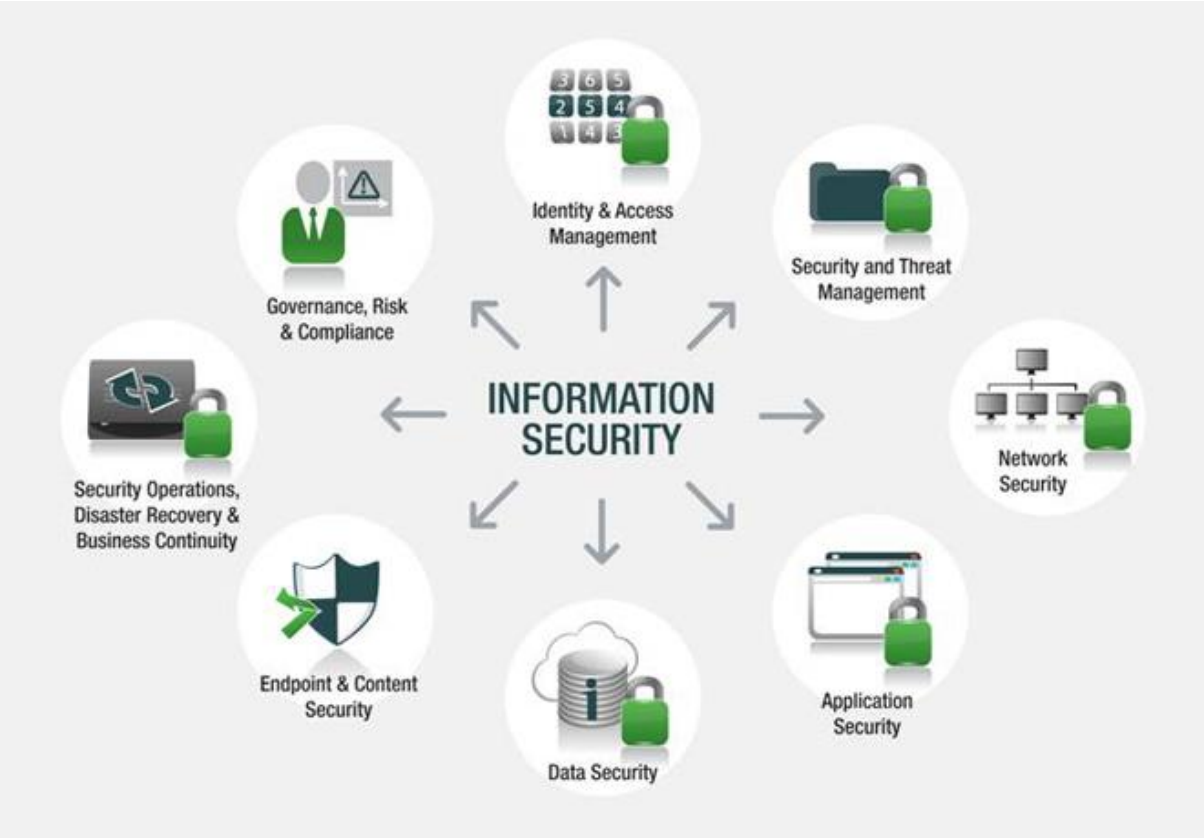

Figura 1 Gli obiettivi principali perseguiti dalla sicurezza Informatica [2]

L'analisi del rischio tipicamente precede la fase di messa in esercizio di un sistema informatico, in modo da poter individuare subito quali contromisure vanno adottate per rendere il sistema piuttosto stabile ed evitare possibili *attacchi*, ovvero lo sfruttamento delle vulnerabilità da parte di un utente malevolo.

1.2. Classificazione degli attacchi informatici

La diffusione di software per attacchi informatici, comunemente definiti *malware*, è in continuo aumento così come riportato da analisi di sicurezza effettuate da società come Symantec [3] e Verizon.
Il rapporto di Symantec evidenzia come gli *hacker* stiano diffondendo malware molto più velocemente rispetto agli scorsi anni, con la creazione, in un anno, di 317 milioni nuovi software malevoli con una media di quasi un milione di malware creati al giorno.
Il *Verizon's 2015 Data Breach Investigations Report* [4] evidenzia, inoltre, come il 90% di questi malware sfruttino debolezze e *bug (difetti)* presenti nella rete e nei sistemi da anni, ma che tuttora non sono stati risolti.

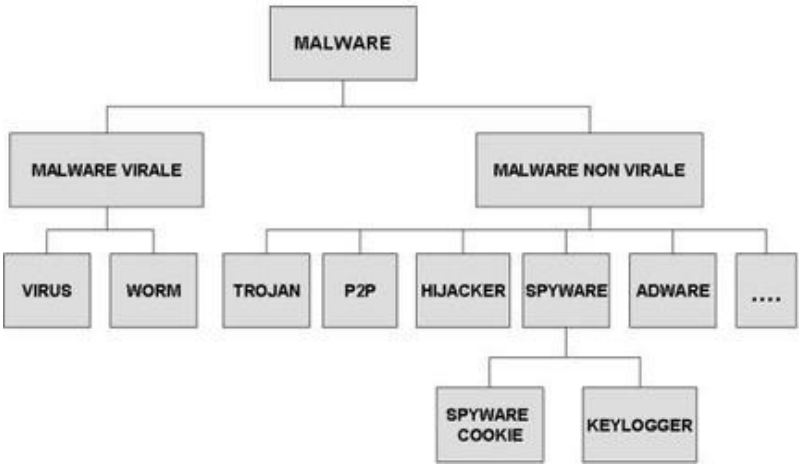

Figura 2 Classificazione dei Malware [5]

In Figura 2 [5] viene illustrata la tassonomia dei malware [6], le cui principali tipologie sono:

1. *Virus*: sono parti di codice che si diffondono copiando all'interno di altri programmi, o in una particolare sezione del disco fisso, una possibile versione evoluta di loro stessi, in modo da essere eseguiti ogni volta che il file infetto viene aperto. Si trasmettono da un computer ad un altro tramite lo spostamento di file infetti ad opera degli utenti;
2. *Worm*: questi malware non hanno bisogno di infettare altri file per diffondersi, perché modificano il sistema operativo della macchina ospite in modo da essere eseguiti automaticamente e tentano di replicarsi sfruttando per lo più Internet. Per indurre gli utenti ad eseguirli sfruttano dei bug di alcuni programmi per diffondersi automaticamente. Il loro scopo è rallentare il sistema con operazioni inutili o dannose;
3. *Trojan Horse*: software che oltre ad avere delle funzionalità "lecite", utili per indurre l'utente ad installarli ed utilizzarli, contengono istruzioni dannose che vengono eseguite all'insaputa dell'utilizzatore. Non possiedono funzioni di auto-replicazione, quindi per diffondersi devono essere trasmessi alle vittime che li eseguono consapevolmente;

4. *Ransomware*: tipologia di malware che limita l'accesso di un sistema al proprio utente. Di solito questi software cercano di forzare gli utenti a pagare, attraverso dei metodi online di pagamento, per riavere i loro dati indietro o per accedere ad alcune funzionalità del loro sistema;
5. *Spyware*: raccolgono informazioni dal sistema su cui sono installati per trasmetterle ad un destinatario interessato. Le informazioni carpite possono andare dalle abitudini di navigazione fino alle password e alle chiavi crittografiche di un utente;
6. *Rootkit*: sono solitamente composti da un driver e a volte, da copie modificate di programmi normalmente presenti nel sistema. I rootkit non sono dannosi di per sé, ma hanno la funzione di nascondere, sia all'utente che ai programmi di protezione, come gli antivirus, la presenza di particolari file o impostazioni del sistema. Vengono quindi utilizzati per mascherare spyware e trojan;
7. *Hijacker*: si appropriano generalmente dei browser per aprire pagine internet indesiderate;
8. *Adware*: programmi software che presentano all' utente messaggi pubblicitari durante l'uso. Possono causare danni quali rallentamenti del pc e rischi per la privacy in quanto comunicano le abitudini di navigazione ad un server remoto;
9. *Backdoor*: consentono di aggirare in parte o completamente le procedure di sicurezza di un sistema informatico. Consentono l'accesso remoto, senza autorizzazione, ad un dispositivo da parte di un utente malintenzionato. Oltre ad essere molto pericolosi per l'integrità delle informazioni presenti sul sistema, le backdoor installate dai virus possono essere utilizzate per condurre degli attacchi di tipo *Denial of Service.*

Queste tipologie di software permettono di attuare degli attacchi che sono generalmente suddivisi in *attivi o passivi.*
I primi compromettono l'integrità e la disponibilità, cioè hanno come obiettivo l'alterazione delle informazioni o il danneggiamento del sistema per renderlo inutilizzabile.
I secondi, al contrario, compromettono l'autenticazione e la riservatezza per entrare in possesso di dati privati.
Malware di tipo attivo sono solitamente i virus e i worm, al contrario, adware e spyware appartengono alla seconda categoria.

1.2.1. Denial of Service (DoS/DDoS)

Questo tipo di attacco ha lo scopo di rendere inutilizzabile un sistema informatico, che offre un servizio ai client, attraverso l'esaurimento delle risorse dello stesso, impedendogli, di conseguenza, di erogare il servizio.
Questo obiettivo viene conseguito attraverso un invio di molti pacchetti di richieste, di solito ad un web server, saturandone le risorse e facendolo diventare indisponibile.

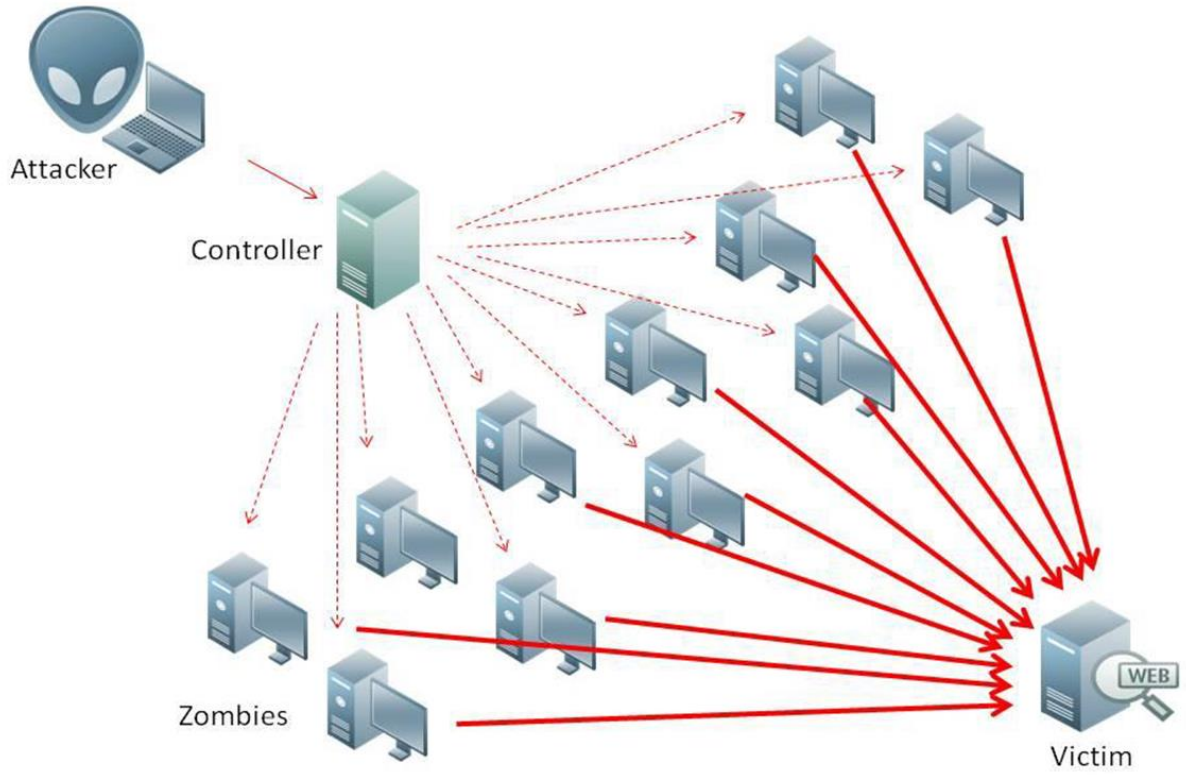

Figura 3 Il funzionamento di un attacco di tipo Dos [7]

Come mostrato in Figura 3 [7], un *hacker*, che abbia intenzione di effettuare questa tipologia di attacco, ricorre all'utilizzo di un numero molto elevato di computer *zombie* che nello stesso istante richiedono un servizio al sistema vittima, saturandone la coda di esaudimento delle richieste e di conseguenza rendendolo indisponibile agli utenti leciti.
I computer utilizzati per questo attacco vengono definiti zombie poiché infettati, tramite un apposito programma trasmesso da uno dei malware analizzati nel precedente paragrafo, dall'utente malevolo, che, assumendone il pieno controllo, è in grado, quindi, di utilizzarli a proprio piacimento senza che gli utenti leciti, proprietari di questi dispositivi, se ne rendano conto.
Questo attacco è particolarmente efficace perché tutti i sistemi in rete sono potenziali vittime e per la impossibilità di limitarli.
Infatti, trattandosi di connessioni apparentemente legittime, è impossibile bloccarle senza interrompere anche il flusso realmente inoffensivo. Limitando drasticamente il numero di sessioni aperte simultaneamente, però, si può ridurre considerevolmente l'impatto dell'attacco senza limitare il flusso dei pacchetti regolari.
Una variante di questo attacco è il *Distributed Denial of Service*, dal funzionamento identico, ma con l'organizzazione dei computer attaccanti in vere e proprie unità che prendono il nome di *botnet.*

1.2.2. DNS Cache Poisoning

Questo tipo di attacco ha come scopo quello di fuorviare i *name server* (o DNS server; si occupano della risoluzione dei nomi dei siti web) permettendo la modifica nell'associazione indirizzo IP/nome del server.
Ciò consente di reindirizzare un nome di dominio web verso un indirizzo IP diverso da quello originale.
La tecnica consiste nell'ingannare un DNS server facendogli credere di ricevere delle informazioni autentiche, quando in realtà sono state create ad arte per modificarne il comportamento. Le informazioni ricevute vengono inoltre conservate nella cache del server per un certo periodo di tempo diffondendo l'effetto dell'attacco agli utenti leciti che comunicano con il name server.

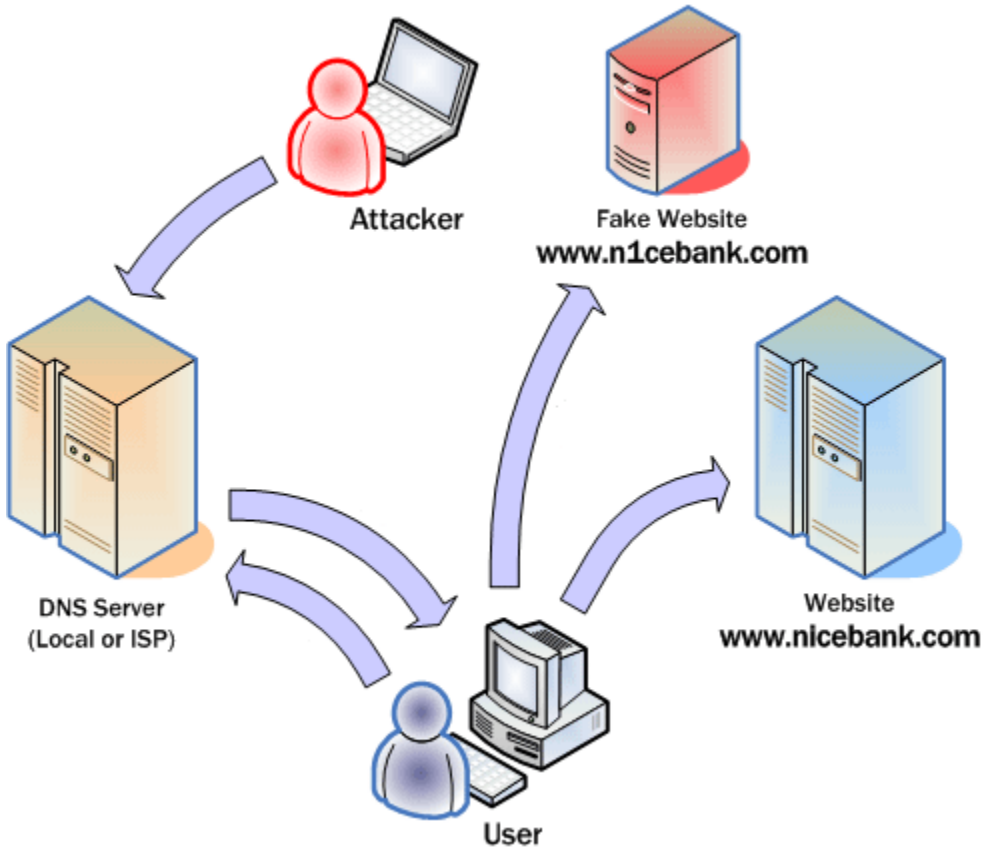

Figura 4 Funzionamento del DNS Cache Poisoning [8]

La Figura 4 [8] mostra il funzionamento di questo attacco. Un utente che richiede l'indirizzo IP di un particolare sito web viene reindirizzato dal DNS server, precedentemente manomesso da un hacker, su un sito fasullo creato ad hoc dall'utente malevolo che tramite questo stratagemma potrebbe acquisire dati sensibili (nell'esempio credenziali bancarie) della vittima.

1.2.3. The Man in the Middle Attack

L'attacco dell'*uomo in mezzo* fa riferimento alla possibilità di un utente malintenzionato di porsi tra due parti comunicanti tra loro e di spiarne la conversazione.
Caratteristica fondamentale è il non permettere che nessuno dei due attori presenti in scena sia in grado di sapere se il collegamento che li unisce reciprocamente sia stato effettivamente compromesso da una terza parte, ovvero un attaccante, in grado così, di osservare, intercettare e replicare verso la destinazione prestabilita il transito dei messaggi tra le due vittime.

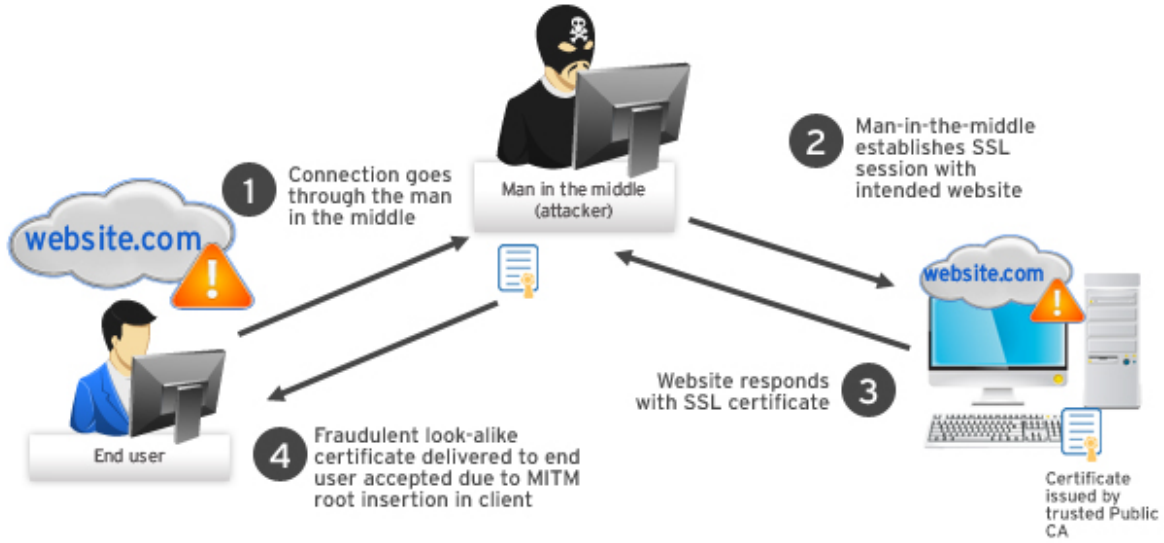

Figura 5 Scenario classico del MITM attack [9]

Come mostrato in Figura 5 [9], l'utente malevolo inganna entrambe le parti in comunicazione manipolando i dati che vengono inviati e ricevuti acquisendo informazioni sensibili dalle vittime.
La crittografia della comunicazione, purtroppo, non garantisce una difesa sempre efficace da questo attacco. Un utente malevolo, infatti, potrebbe trovarsi nel mezzo della comunicazione tra i due utenti legittimi nel momento dello scambio delle chiavi crittografiche che, una volta ottenute, gli consentirebbero di leggere i messaggi, comunque cifrati, che i due attori in scena si stanno scambiando.
Una modalità di difesa largamente utilizzata per prevenire questo attacco è l'utilizzo di chiavi firmate garantite da una terza parte di fiducia, di solito una Autorità Certificante, che ne assicura l'autenticità.

Tuttavia, la noncuranza da parte delle autorità di certificazione nel dare la loro approvazione alla corrispondenza tra le informazioni sull'identità e le relative chiavi pubbliche sono un problema di questi sistemi.

1.3. Principali metodologie di difesa

Data la natura così varia degli attacchi informatici, negli anni si sono diffusi molti sistemi software e/o hardware atti a prevenirli.
Le tipologie più famose [10] sono:

1. *Antivirus:* consentono di proteggere i personal computer dai virus. Devono essere costantemente aggiornati per essere efficaci;
2. *Antispyware:* facilmente reperibili sul web, si occupano della rimozione degli spyware;
3. *Firewall:* garantisce un sistema di controllo degli accessi verificando tutto il traffico che lo attraversa. Protegge contro aggressioni provenienti dall'esterno e blocca eventuali programmi presenti sul computer che tentano di accedere ad internet senza il controllo dell'utente;
4. *Honeypot:* è un sistema o componente hardware o software usato come "trappola" o "esca" a fini di protezione contro gli attacchi di pirati informatici. Solitamente consiste in un computer o un sito che sembra essere parte della rete e contenere informazioni preziose, ma che in realtà è ben isolato e non ha contenuti sensibili o critici; potrebbe anche essere un file, un record, o un indirizzo IP non utilizzato;
5. *Intrusion detection system (IDS):* è un dispositivo software o hardware (a volte la combinazione di tutti e due) utilizzato per identificare accessi non autorizzati ai computer. Le intrusioni rilevate possono essere quelle prodotte da cracker esperti, da tool automatici o da utenti inesperti che utilizzano programmi semiautomatici. Gli IDS vengono utilizzati per rilevare tutti gli attacchi alle reti informatiche e ai computer;
6. *Steganografia:* ha l'obiettivo di mantenere nascosta l'esistenza di dati a chi non conosce la chiave atta ad estrarli;
7. *Firma digitale, Crittografia:* permette di codificare un messaggio in modo tale che solo il destinatario possa accederne al contenuto.

1.3.1. Il protocollo di sicurezza multilivello (MLS)

I sistemi informatici vengono di solito suddivisi in base all'impiego di 3 diverse modalità [11]:

1. *Dedicated mode*: tutti gli utenti hanno libero accesso alle informazioni su un determinato sistema. Il dispositivo è completamente libero da restrizioni di tipo software; possono essere presenti però blocchi di tipo hardware;

2. *System high mode:* tutti gli utenti hanno i permessi per accedere alle informazioni presenti nel sistema ma non tutti hanno la necessità di conoscerle. Viene quindi implementata una strategia di gestione degli accessi tipica dei sistemi multi-utente, in cui viene garantita la disponibilità delle sole informazioni realmente indispensabili per ogni tipologia di utente;
3. *Multilevel mode:* la disponibilità di alcune informazioni è limitata solo ad alcune categorie di utenti con privilegi speciali. Vengono utilizzate tecniche di controllo per l'accesso al sistema e ai file con particolari restrizioni.

Il modello di sicurezza multilivello, che si avvale della terza modalità operativa, viene impiegato per garantire l'accesso autorizzato a delle informazioni solo agli utenti che dispongono dei permessi per visualizzarle.
Viene di solito utilizzato negli enti governativi, militari e di intelligence in cui la riservatezza delle informazioni è fondamentale.
Il termine multilivello si riferisce ai diversi livelli di sicurezza con cui vengono classificate delle informazioni in un sistema, così come mostrato nella figura 6 [12].

Figura 6 I diversi livelli di sicurezza [12]

Questo sistema di sicurezza viene generalmente implementato seguendo due modelli, le cui caratteristiche vengono illustrate nei paragrafi successivi.

1.3.1.1. Il modello Bell – LaPadula

Sviluppato da David Elliot Bell e Leonard J. LaPadula, il modello Bell – LaPadula [13] ha come obiettivo principale quello della confidenzialità controllando gli accessi alle informazioni classificate.
In questo modello i programmi e i processi, chiamati *subjects*, tentano di trasmettere informazioni alle risorse del sistema, chiamate *objects*, attraverso file o dispositivi I/O.
Ogni *subject o object* possiede una etichetta che ne identifica il livello di sicurezza che permette l'interazione con le informazioni trasportate solo secondo delle precise norme.
Le regole ferree su cui si basa questo modello sono:

1. Proprietà di sicurezza semplice: stabilisce che un processo eseguito al livello k di sicurezza può leggere solo oggetti del suo livello o di livelli inferiori (*no read up)*;
2. Proprietà * (chiamata anche proprietà "star"): questa regola impone che un processo eseguito a un livello di sicurezza può scrivere solo oggetti al suo livello o a un livello superiore (*no write down*);
3. Proprietà di sicurezza discrezionale: utilizza una matrice per gli accessi per il controllo degli stessi; indica la possibilità da parte degli amministratori del sistema di modificare le regole di accesso al sistema.

Un esempio di questo modello è raffigurato nella Figura 7. Si può osservare come un processo etichettato come "segreto" abbia permesso di sola scrittura per file di livello pari al suo o superiore e di sola lettura per file di un livello di sicurezza più basso.

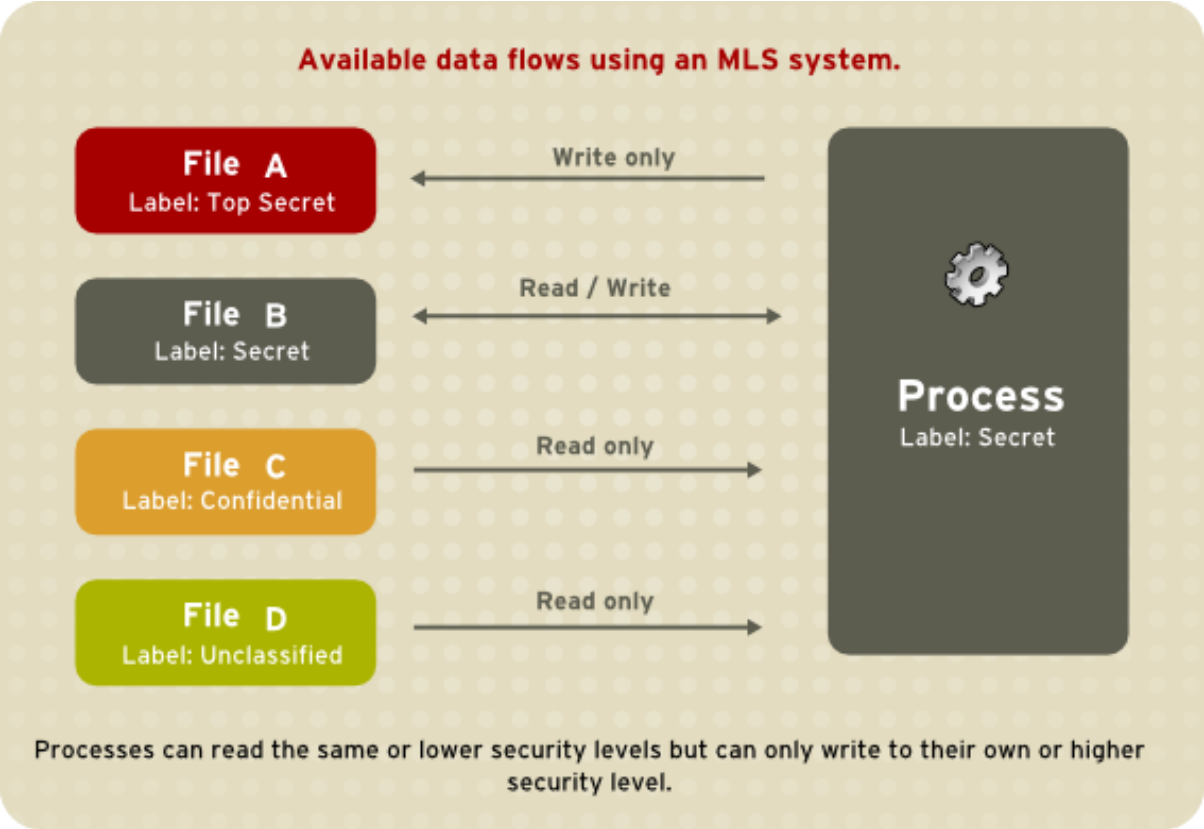

Figura 7 esempio di Modello Bell – LaPadula [12]

Concedendo la libertà di modifica delle regole di accesso ad un amministratore, la terza proprietà rende possibile ad un utente malevolo in grado di "infiltrarsi" nel sistema, con l'uso di un Trojan Horse per esempio, di fare lo stesso mettendo così a repentaglio la sicurezza dell'intero sistema.
Per evitare questo alcuni modelli rendono più restrittiva la proprietà "star", definita di conseguenza come "strong star", che permette la scrittura solo di oggetti del proprio livello, eliminando la possibilità di interazione con livelli superiori.

1.3.1.2. Il modello Biba

Sviluppato nel 1975 da Kenneth J. Biba, il modello Biba [13] ha come obiettivo primario quello di mantenere l'integrità dei dati, prevenendone la corruzione da processi con livello di sicurezza inferiore.
Questo obiettivo viene perseguito seguendo 3 norme, di cui le prime 2 sono l'opposto delle regole presentate dal modello Bell – LaPadula, che sono:

1. Proprietà di integrità semplice: è proibito ad un utente con un determinato livello di integrità di leggere risorse di livello inferiore (*no read down*);
2. Proprietà * (o star) di integrità: non è permessa la scrittura di *objects* che hanno un livello di integrità superiore rispetto al *subject* in uso (*no write up*);
3. Proprietà di invocazione: un processo di integrità basso non può richiedere l'accesso ad un livello superiore.

Il funzionamento generale del modello è riassunto in Figura 8 [14], che mostra le interazioni tra processi e oggetti permesse da un sistema che segue questo schema.

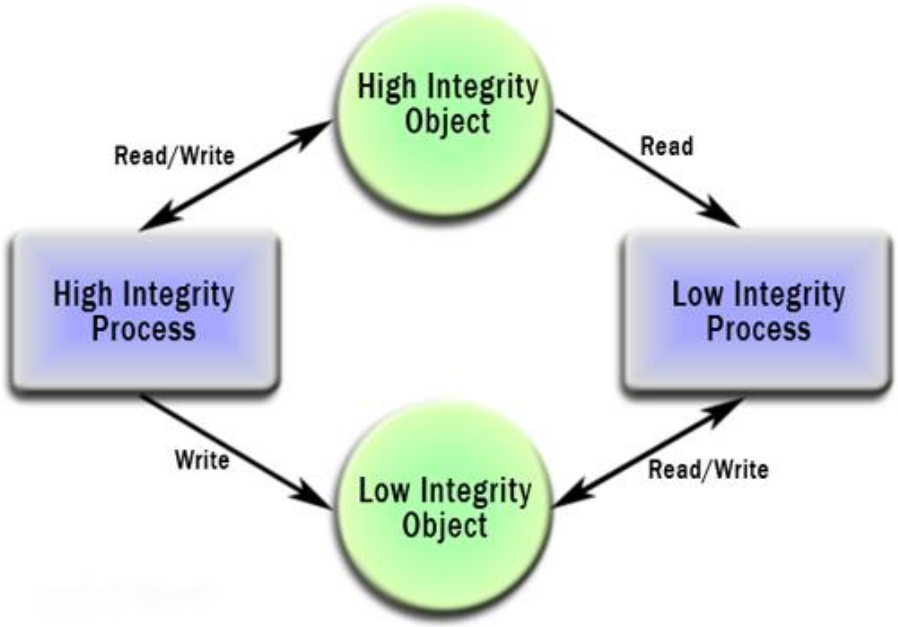

Figura 8 Classico funzionamento del modello Biba [14]

1.4. Sicurezza nei sistemi mobili

Di fondamentale importanza è comprendere l'impatto di questi attacchi nel campo mobile.
La diffusione capillare degli smartphone ha spinto milioni e milioni di utenti a custodire informazioni sensibili sul proprio dispositivo mobile così come riportato in uno studio compiuto da *Dimensional Research* [15] di cui una delle statistiche più rilevanti viene illustrata in Figura 9.

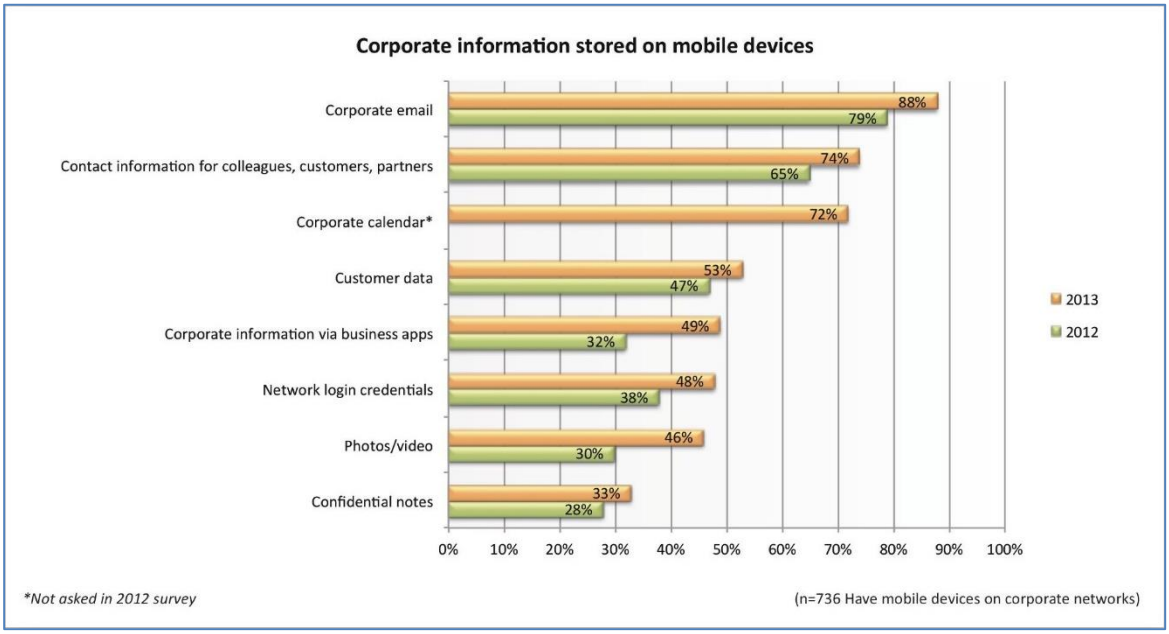

Figura 9 Informazioni salvate su un dispositivo mobile [15]

Il grafico mostra le elevate percentuali di informazioni riguardanti il campo lavorativo, come quello di una azienda, che vengono memorizzate in un dispositivo mobile, evidenziando un aumento rilevante rispetto agli stessi dati raccolti nell'anno precedente.
Questo ruolo di nuove banche dati da parte dei dispositivi mobili li rende dei bersagli ambiti da parte degli utenti malintenzionati che hanno come fine quello di impadronirsi di informazioni private della vittima o di rubarne l'identità o semplicemente di negargli il servizio.
Gli attacchi sfruttano delle debolezze degli smartphone che derivano sia dalle diverse metodologie di comunicazione come gli SMS (*short messaging service)*, gli MMS (*multimedial messaging service*), il Wi-Fi e il bluetooth ma anche vulnerabilità correlate al sistema operativo, al web browser e alla diffusione dei malware.
Le diverse tipologie di attacchi verranno analizzate in dettaglio nei paragrafi successivi.

1.4.1. Attacchi tramite SMS/MMS

Questi attacchi sono efficaci solo su determinate marche di smartphone in cui è presente una implementazione non ottimale del servizio di messaggistica.

L’invio di SMS che sfruttano queste vulnerabilità possono provocare un *denial of service* dello smartphone che riceve questi messaggi.
Inviando un messaggio con contenuto binario ad un particolare dispositivo NOKIA se ne può, per esempio, provocare il blocco con conseguente riavvio dello stesso.
Le offensive tramite MMS sfruttano il contenuto multimediale di questa modalità di comunicazione, inserendo un virus nell’elemento multimediale, infatti, si può infettare un dispositivo mobile qualora l’utente cercasse di visualizzarlo.

1.4.2. Attacchi tramite Wi-Fi e Bluetooth

Gli attacchi che sfruttano la connessione senza fili sono particolarmente pericolosi perché possono causare l’infezione dell’intero network della vittima oltre a quello dello smartphone.
Infatti, i dispositivi mobili implementano alcune funzionalità come quella del salvataggio della rete, ovvero una volta inserite le credenziali del network il sistema le può memorizzare e inserire in maniera automatica al prossimo accesso, che potrebbero permettere ad un utente malevolo, che abbia preso controllo del dispositivo mobile, di spiare i dati in uscita dalla rete della vittima qualora non fossero criptati.
Queste offensive si basano sulla debolezza della chiave con cui viene criptata la rete senza fili. Nel corso degli anni però sono stati fatti molti progressi in questo campo, col passaggio dalle chiavi WEP (brevi e non crittate) alle WPA (con l’introduzione dell’*encrypt* dinamico) prima, e alle WPA2 dopo, che hanno reso un attacco di questo tipo piuttosto complicato per un hacker.
Gli attacchi tramite bluetooth si basano sull’invio di un file infetto ai dispositivi in modalità “visibile” (rilevabile da una ricerca di un qualsiasi dispositivo bluetooth). Nel caso in cui l’utente accettasse questo trasferimento, il telefono verrebbe infettato dal virus presente nel file.
Queste offensive si basano quindi molto sull’imprudenza del possessore del dispositivo mobile e con le giuste precauzioni possono essere facilmente evitati.

1.4.3. Attacchi basati sul sistema operativo

Questa tipologia di aggressione si basa sulle debolezze strutturali del sistema operativo del dispositivo mobile. Recentemente è stato scoperto un nuovo bug di Android che permette di avere l’accesso fisico al dispositivo tramite un particolare utilizzo del *lock screen* [16].
Questa tipologia di attacchi possono essere molto efficaci e dannosi, dall’altro lato della medaglia però, le case produttrici dei sistemi operativi risolvono velocemente questi bug una volta manifestati, rendendo l’offensiva inutilizzabile dopo un breve periodo di tempo.

1.4.4. Attacchi tramite malware

Nel campo mobile, così come quello dei dispositivi fissi, un gran numero di infezioni viene raggiunto tramite l'utilizzo di software malevoli implementati da hacker direttamente per lo scopo.
Come riportato da uno studio di *Information Week* [17], il numero dei malware per Android è raddoppiato negli ultimi sei mesi.
I malware che sono in grado di infettare un dispositivo mobile sono delle stesse tipologie analizzate nel paragrafo 1.2.
Con l'avvento delle applicazioni (programmi eseguibili su dispositivi mobili), i malware hanno trovato un nuovo mezzo trasmissivo. Gli hacker, infatti, possono implementare delle applicazioni come veri e propri Trojan Horse, fornendo delle funzionalità che invogliano l'utente ad installarle sul proprio dispositivo, accedendo ai dati sensibili della vittima tramite il consenso al trattamento degli stessi che quasi la totalità delle applicazioni "lecite" richiede e che un utente imprudente accetta senza esitazione.
Dopo aver passato al vaglio le più note tipologie di attacchi a dei sistemi mobili, nel prossimo paragrafo vengono esaminati i principali metodi attuati per difendere questi dispositivi da possibili infezioni.

1.5. Contromisure e strategie di difesa

Le metodologie impiegate per la difesa dei dispositivi mobili sono molto varie.
Come già introdotto, molte delle infezioni in cui un uno smartphone può incorrere, nella maggior parte dei casi, sono dovute più ad una negligenza da parte della vittima più che alla bravura dell'hacker.
Risulta quindi fondamentale che un utente sia a conoscenza dei rischi a cui può andare incontro, cercando di prevenirli piuttosto che risolverli in un secondo momento.
Nei paragrafi successivi vengono esaminati in dettaglio i vari livelli di sicurezza che vengono implementati in questi dispositivi.

1.5.1. Sicurezza nei sistemi operativi mobili

I sistemi operativi dei dispositivi mobili applicano il principio di *sandbox* [18]. Con questo termine si indica la divisione netta tra i vari processi presenti in un apparecchio mobile, in modo tale che non ci siano interferenze e che non si possano danneggiare tra di loro.
Questo principio risulta particolarmente efficace nell'ambito della sicurezza perché permette di limitare l'area di infezione che un malware potrebbe essere in grado di colpire.
Il *sandboxing* viene implementato in maniera differente a seconda dell'OS, per esempio iOS (sistema operativo mobile di Apple) viene limitato l'accesso alle API alle applicazioni

pubblicate sull'App Store (market virtuale delle applicazioni di Apple), mentre Android limita l'accesso a determinati dati dell'apparecchio.
Le principali funzionalità di difesa dell'OS [18] vengono di seguito descritte:

- *Rootkit Detectors*: l'infezione da parte di un rootkit è pericolosa in ambito mobile quanto in quello desktop. Tramite questi malware si può ottenere il controllo parziale o totale del dispositivo con la conseguente possibilità di avere accesso totale alle impostazioni e ai dati dello stesso. Per fare un esempio di come funzionino questi rilevatori si può citare la contromisura chiamata *Chain of Trust* di iOS [19]. Le applicazioni necessarie per il boot del dispositivo devono avere una speciale firma certificata da Apple. Nel caso in cui un errore fosse riscontrato il sistema smetterebbe di funzionare rendendolo inutilizzabile ad un utente malevolo.
- Isolamento dei processi: Android associa ad ogni processo un identificatore (metodologia ereditata da Linux su cui Android si basa). Il sistema operativo, tramite un check di questo identificatore, assegna una zona della sandbox al processo, proibendogli, di conseguenza, di interferire con altre applicazioni o con le proprietà del sistema.
- Permessi sui file: in Android è anche presente un controllo sui file. Un processo non può modificare i file che non gli competono evitandone così anche la corruzione.
- Protezione della memoria: un processo non può accedere alle aree di memoria allocate per altri processi.

1.5.2. Software per la protezione

Come per i dispositivi fissi anche per la sicurezza mobile sono stati sviluppati software in grado di monitorare lo stato dell'apparecchio proteggendolo da eventuali attacchi.
Oltre ad antivirus e firewall, già analizzati nell'ambito desktop, vengono impiegati altri strumenti [18] quali:

- Notifiche visuali: informano l'utente di attività in esecuzione sul dispositivo in modo da insospettirlo qualora ci siano in funzione processi non richiesti;
- Turing test: ha lo scopo di chiedere una conferma all'utente prima di eseguire un'azione mediante la richiesta di un codice, spesso visualizzato tramite *captcha,* che permette di verificare che l'utente lecito stia effettivamente utilizzando il dispositivo e che non sia stato un sistema automatizzato (come un malware) ad eseguire il processo.
- Identificazione biometrica: impiega il riconoscimento delle caratteristiche morfologiche di un utente (occhi, impronte digitali ecc.) per verificarne l'identità.

Dopo aver introdotto i principali concetti riguardanti la sicurezza informatica, nel prossimo capitolo verranno esaminati i covert channel, argomento centrale di questa tesi, che sono una particolare tipologia di attacco informatico, il cui principale punto di forza è la difficoltà di

rilevamento che li rendono uno delle offensive più pericolose che si possano realizzare e di conseguenza meritano una disamina più approfondita per valutarne le potenzialità e i rischi.

2 I Covert Channel

Il termine *covert channel*, per la prima volta introdotto da Lampson nel 1973 [20], indica un particolare attacco informatico in grado di contravvenire al principio di sicurezza multilivello, descritto nel capitolo 1, e di permettere quindi ad un utente malintenzionato di osservare e sfruttare, a proprio vantaggio, delle attività di altri utenti.
Un covert channel, infatti, permette una trasmissione non rilevabile tra processi con diversi livelli di sicurezza, assolutamente proibito dal modello di sicurezza multilivello, tramite la manipolazione delle risorse (timing degli eventi, memoria ecc.) di un dispositivo, trasmettendo, quindi, dati illeciti in maniera celata ad un eventuale organo di controllo.
Risulta quindi fondamentale studiare questo tipo di attacco in modo da, se non eliminare completamente la possibilità di instaurazione di questo canale nascosto, limitarne l'effettivo funzionamento.

2.1. L'analogia dei due prigionieri

Nel 1983 Simmons [21] descrisse un covert channel secondo l'analogia dei due prigionieri, che ben illustra l'obiettivo di un canale nascosto.

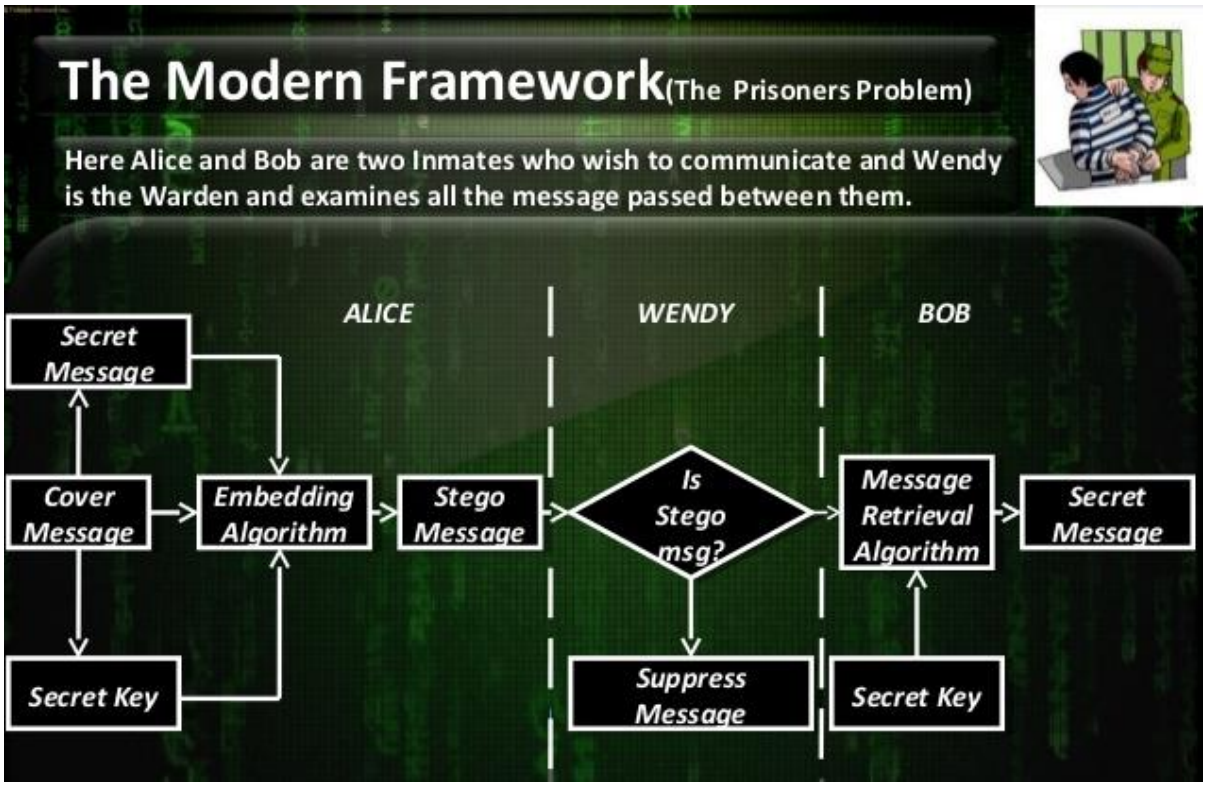

Figura 10 Il problema dei due prigionieri [22]

Alice e Bob sono due prigionieri che vogliono evadere da una prigione, pianificando la loro fuga insieme. I due galeotti però non possono comunicare in maniera libera per la presenza di Wendy, il guardiano della prigione, in grado di leggere i messaggi che i due prigionieri si scambiano e che li potrebbe spostare in due celle di isolamento se venisse a conoscenza dei loro piani o se sospettasse di una comunicazione criptata tra i due.
Estendendo il concetto alla comunicazione tra dispositivi, Alice e Bob sono due host che vogliono scambiarsi informazioni in maniera celata, in modo tale da non insospettire Wendy che rappresenta un eventuale organo di controllo della rete.
In Figura 10 viene illustrato un esempio di comunicazione tra i due host. Alice codifica un messaggio segreto in un messaggio lecito e cerca di inviarlo a Bob. Wendy controlla il messaggio e nel caso in cui sospetti la presenza di un messaggio segreto provvede a sbarazzarsene. Al contrario, se l'informazione passa il controllo del guardiano viene consegnato al destinatario reale, ovvero Bob.
L'esempio fa riferimento ad una comunicazione classica, che non segue i principi di un canale nascosto in cui l'informazione illecita viene trasmessa attraverso la manipolazione di alcune risorse di un sistema, eludendo, di conseguenza, il controllo del guardiano.
Un covert channel è quindi il mezzo attraverso cui i due prigionieri possano comunicare senza essere notati, un metodo di trasmissione di informazioni non convenzionale, che sfrutta metodologie di comunicazione lecite per inviarne di illecite, nascondendo l'esistenza stessa di una trasmissione non autorizzata.

2.2. Tipologie di Covert Channel

I covert channel vengono di solito suddivisi in due grandi sottocategorie in base al modo in cui celano l'informazione illecita:

1. *Covert Storage Channels:* celano le informazioni illecite in aree di memoria accessibili da un receiver;
2. *Covert Timing Channels:* manipolano il tempo che intercorre tra un datagramma e il suo successivo per inviare informazioni binarie.

Il covert channel oggetto di questa tesi è di tipo *timing*.
I covert channel di questa tipologia, infatti, rappresentano una minaccia maggiore a causa della loro rilevazione molto più difficoltosa rispetto a quella di un *covert storage channel* e per l'assenza di soluzioni definitive al problema, che rendono questo tipo di attacco un argomento di alto interesse.
Entrambe le tipologie di covert channel verranno comunque analizzate in dettaglio nei paragrafi seguenti.

2.2.1. Covert Storage Channels

Come precedentemente introdotto, questo tipo di covert channel basa il suo funzionamento sull'inserimento di informazioni illecite in aree di memoria condivise o accessibili in un

secondo momento dal receiver. All'interno della tassonomia di questi canali nascosti c'è una ulteriore suddivisione in:

1. *Local Covert Storage Channel;*
2. *Remote Covert Storage Channel.*

I canali nascosti, cosiddetti "locali", sono quei canali in cui la risorsa di memoria è condivisa sullo stesso sistema e può essere direttamente o indirettamente acceduta da un utente o da una risorsa locale.
Un noto esempio di questo tipo di canale nascosto è denominato *the disk arm channel* [23], che sfrutta il posizionamento dei settori di un disco rigido per inviare dei dati binari in maniera celata, ovvero modifica le priorità di esaudimento delle richieste del disco in modo da far posizionare il braccio meccanico su un determinato settore per inviare un bit covert pari a 0 o pari ad 1 a seconda delle esigenze.
I *remote covert storage channel*, al contrario, sono dei canali nascosti tra due entità che comunicano attraverso una qualsiasi rete di comunicazione.
Il loro funzionamento si basa sull'inserimento di messaggi illeciti nei campi dei pacchetti IP mostrati in figura.

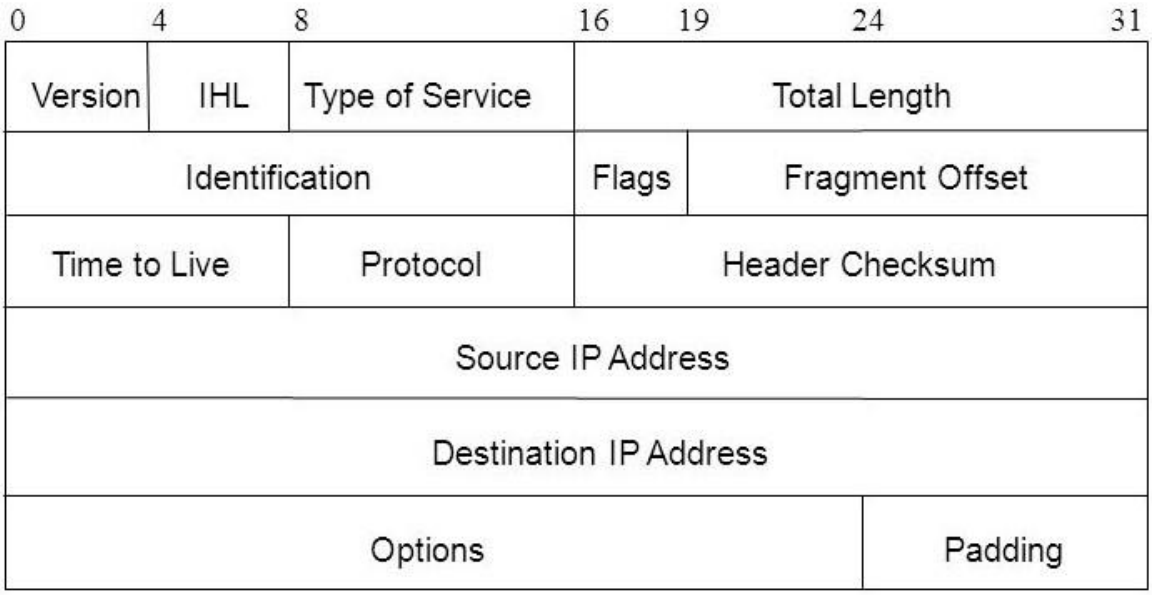

Figura 11 Classici campi di un pacchetto IP [24]

I campi maggiormente utilizzati per lo scopo sono quelli del *Type of Service* o dell'*Header Checksum*.
Studi approfonditi su questo tipo di canali nascosti sono stati pubblicati in un articolo da Craig Rowland [25].
L'autore ha dimostrato che è possibile utilizzare per l'instaurazione di questo covert channel anche due campi tipici del protocollo di rete TCP che sono:

1. Sequence number;
2. Acknowledgment number.

Sebbene sia possibile comunicare facilmente attraverso questa schema, Rowland ha messo in discussione l'effettiva robustezza di questo canale nascosto, evidenziando come tale implementazione sia rilevabile nel caso in cui vengano inviati più pacchetti con lo stesso *sequence number*. Inoltre *un firewall con filtro dei pacchetti potrebbe interrompere il traffico nel caso in cui vengano utilizzati degli indirizzi di rete reali* [25].
Gli *storage covert channel* sono quindi un metodo di comunicazione nascosto efficiente ma rilevabile con meno sforzi rispetto ai canali basati sul *timing*, che vengono analizzati nel paragrafo successivo.

2.2.2. Covert Timing Channels

I covert channel basati sul timing, al contrario di quelli basati sulla condivisione di risorse, sono molto più difficili da rilevare ma meno precisi.
La causa di questo inconveniente è da cercare nella natura stessa di un *covert timing channel.*
La comunicazione tramite la rete, infatti, introduce sempre sia del rumore nel canale di comunicazione sia dei ritardi, fattori che limitano le prestazioni di questa modalità di trasmissione illecita.

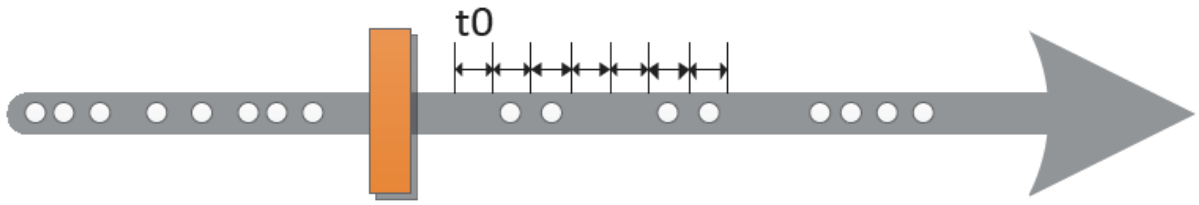

Figura 12a Funzionamento di un Remote Storage Covert Channel [26]

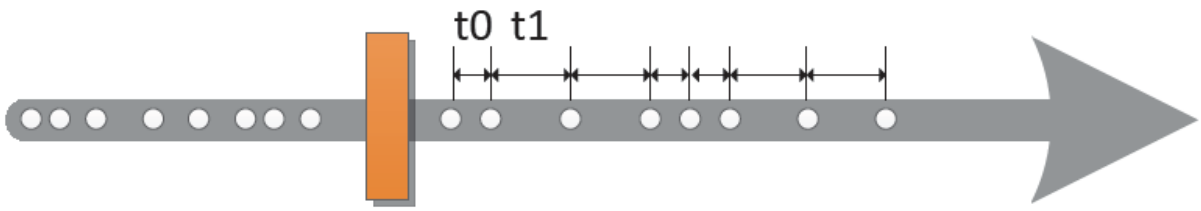

Figura 12b Funzionamento di un Timing Covert Channel [26]

Le Figure 12a e 12b [26] illustrano la diversa natura delle due tipologie di canali nascosti. Nel caso di un covert channel di tipo storage non c'è nessuna manipolazione della temporizzazione, di conseguenza i pacchetti possono essere inviati in intervalli di tempo molto variabili. Al contrario nel canale nascosto di tipo timing si può notare come l'invio dei datagrammi debba avvenire in maniera piuttosto regolare; variazioni significative

dell'intervallo causate da ritardi della rete, quindi, possono inficiare la comunicazione nascosta.
Questo tipo di covert channel viene suddiviso in altre due sottocategorie:

1. *Active Covert Timing Channel*: genera del traffico internet per inviare il messaggio nascosto;
2. *Passive Covert Timing Channel*: sfrutta il traffico esistente nella rete per inviare il messaggio covert, sono i più difficili da individuare ma in generale meno performanti dei canali "attivi".

La principale contromisura a questo tipo di attacco è l'implementazione di *timing jammers* [27], ovvero degli intermediari nella rete in grado di modificare il tempo di arrivo dei pacchetti.
Questa funzionalità però non è molto efficace, infatti, non ha come fine quello di rilevare la presenza di un canale nascosto ma bensì di limitarne gli effetti.
Un utente malevolo, conscio della presenza di questi *jammers*, potrebbe aggirare questa forma di difesa aumentando l'intervallo di tempo tra un datagramma e il successivo, riducendo di conseguenza la banda del canale, in modo tale da rendere insignificante la modifica introdotta dal *jammer.*
La rilevazione di queste comunicazioni illecite risulta quindi sempre la *best practice* da seguire, in modo da poter anche identificare e localizzare gli utenti malevoli che si avvalgono di questo tipo di attacco.
Il covert channel realizzato in questa tesi si è avvalso di un famoso algoritmo, presentato per la prima volta da Padlisky [28] che viene descritto nel dettaglio nel paragrafo 2.2.2.1.

2.2.2.1. The ON/OFF Scheme

Questo schema è uno dei più famosi quando si parla di *covert timing channel*.
Presentato da Padlisky, come già introdotto, e realizzato anche da Cabuk [27], basa il suo funzionamento sulla presenza o assenza di comunicazione in un dato intervallo temporale, concordato a priori tra sender e receiver.
L'informazione che viene inviata tramite questo algoritmo è di tipo binario, ed il funzionamento dello stesso viene ora illustrata in dettaglio (nel caso di *active covert timing channel).*

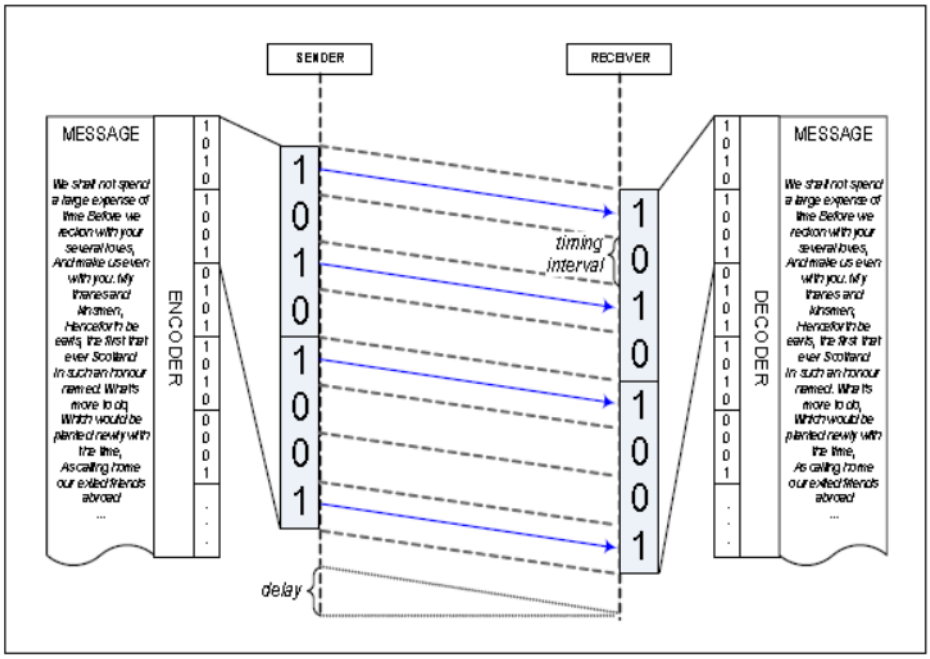

Figura 13 Trasmissione secondo l'ON/OFF scheme [27]

Come riportato nella figura 13, sono presenti un sender e un receiver che vogliono comunicare attraverso un *covert timing channel*, che gli consenta di trasmettere un determinato messaggio in maniera celata.
Per raggiungere questo scopo, il sender deve codificare l'informazione illecita che vuole trasmettere in una sequenza di valori binari ed operare secondo la seguente procedura:

1. Se nel messaggio illecito codificato in binario c'è un bit di valore 1 allora si può trasmettere un datagramma contenente informazione lecita (rappresentato in figura dalle frecce di colore blu);
2. Se il bit covert che si vuole inviare è di valore 0, il sender si pone in un periodo di silenzio pari al *timing slot* concordato con il receiver (rappresentato in figura dalle frecce tratteggiate).

Il receiver monitora ogni intervallo di tempo, determinando se un datagramma è giunto durante il *time slot* concordato o meno. La *detection* di un pacchetto in quell'arco di tempo implica la decodifica di un bit nascosto di valore 1. Al contrario, la mancata rilevazione di un pacchetto nel periodo pattuito porta ad una decodifica di un bit di valore 0.
Il risultato è quindi una sequenza di valori binari che il destinatario è in grado di interpretare, ricostruendo l'informazione illecita che il mittente ha trasmesso.
In questo modello non c'è una effettiva trasmissione di informazioni illecite, ed è proprio questo fattore che rende questo canale nascosto molto pericoloso. Il messaggio viene inviato mediante manipolazione delle risorse "pulite" presenti nella rete rendendo particolarmente arduo, se non addirittura impossibile, ad un organo di controllo, ignaro di questa procedura, di rilevare questo tipo di trasmissione, così come riportato in figura 14 nel caso di un *firewall*, che viene facilmente aggirato da questa metodologia di comunicazione.

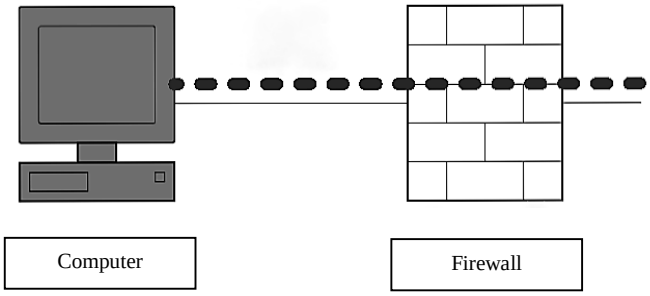

Figura 14 Capacità dei covert timing channel di passare inosservati a organi di difesa

La temporizzazione degli eventi è allo stesso tempo il principale punto di forza e di debolezza di questo algoritmo, infatti, se questa procedura permette una comunicazione "trasparente" tra mittente e destinatario, è anche vero che la possibilità di una decodifica errata è elevata.
Questa problematica è dovuta al rumore che ogni canale di comunicazione introduce così come i ritardi dovuti all'uso della rete nel momento della trasmissione, fattori non prevedibili a priori e che possono inficiare l'interpretazione dello *stream* dei bit covert da parte del receiver.

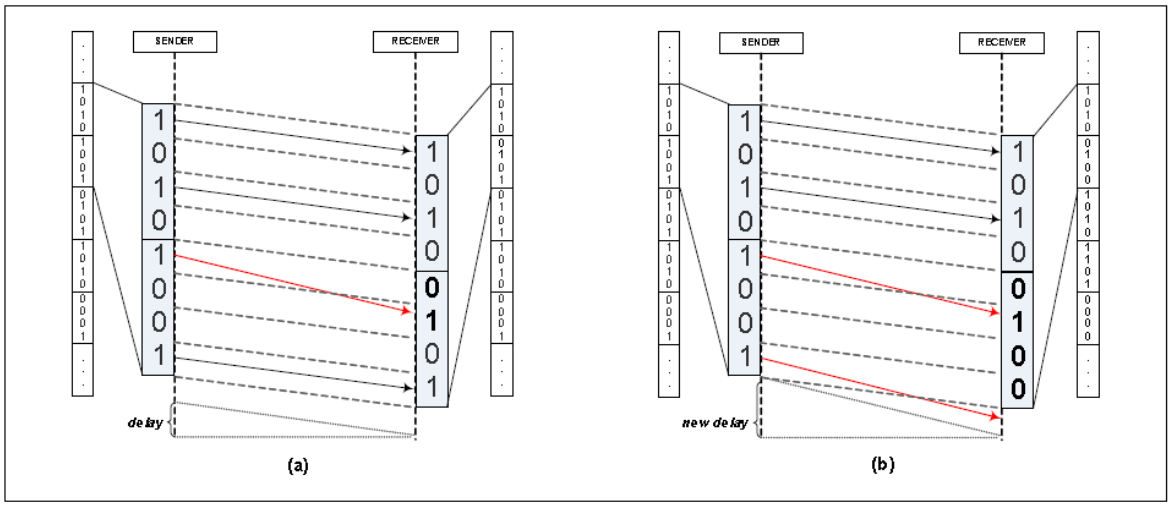

Figura 15 Problemi di sincronizzazione nell' ON/OFF scheme [27]

Come riportato nella figura 15, un ritardo potrebbe provocare una interpretazione sbagliata da parte del destinatario, aggiungendo un bit errato alla sequenza originale, provocandone uno *shift* in grado di compromettere l'intera serie di valori binari.
Timing slot più ampi possono rendere meno cospicuo l'effetto del ritardo introdotto dalla rete ma con un conseguente sacrificio della banda del canale.
Risulta fondamentale, quindi, integrare a questo algoritmo delle procedure di sincronizzazione che possano migliorarne le prestazioni, metodologie che verranno analizzate nel capitolo 3, dedicato appunto alla realizzazione di un canale nascosto.

Lo schema ON/OFF ha trovato negli anni diverse implementazioni rispetto a quella qui descritta.
Per citarne alcune, Esser [29] ha infatti realizzato un covert channel attraverso un web server, che modulava il ritardo con il quale inviava le risposte al client seguendo così lo schema finora descritto.
Un'altra implementazione è quella di Murdoch [30] che ha sfruttato il *rate* di trasmissione dei pacchetti e il loro *timestamp*.
Questa realizzazione necessita di un intermediario che riceve i pacchetti per poi trasmetterli al destinatario reale del covert channel. Il canale implementato da Murdoch sfrutta il fatto che la temperatura della CPU è influenzata dal numero di richieste che vengono eseguite per unità di tempo. Il sender, inviando molte richieste all'intermediario o rimanendo silente per un determinato intervallo, provoca la variazione di temperatura della CPU dell'host che fa da tramite tra mittente e destinatario, modificandone di conseguenza anche il valore del clock.
Il receiver, quindi, osserva le variazioni del timestamp dei datagrammi, introdotte dall'intermediario, decodificando il messaggio celato.
Nel capitolo che segue, dedicato alla implementazione, ci sarà una analisi più approfondita delle debolezze strutturali dei *covert timing channel* descritte finora, con la disamina delle soluzioni adottate durante la progettazione.

3 Implementazione di un Timing Covert Channel

Nei capitoli precedenti sono stati introdotti i concetti di sicurezza informatica e in dettaglio quello di covert channel. In questo capitolo verrà invece analizzata la parte progettuale di questa tesi ovvero l'implementazione di un timing covert channel in ambiente Android, sistema operativo per dispositivi mobili di proprietà di Google.inc basato su Kernel Linux. Adotta una licenza *open source* che permette, a coloro che siano interessati, di modificare liberamente il codice sorgente.

3.1. Scopi ed obiettivi

Il progetto prevede la realizzazione di un covert channel in ambiente Android in grado di inviare un messaggio in maniera totalmente celata ad un eventuale organo di controllo, analizzandone prestazioni e costi.
La scelta di realizzare e analizzare un canale nascosto su Android è stata dettata dall'enorme popolarità e diffusione di questo OS.
Secondo le stime della *International Data Corporacy* [31], i cui grafici vengono sotto riportati, Android detiene l'82% del mercato dei sistemi operativi mobili, con un incremento considerevole di anno in anno a scapito della concorrenza.

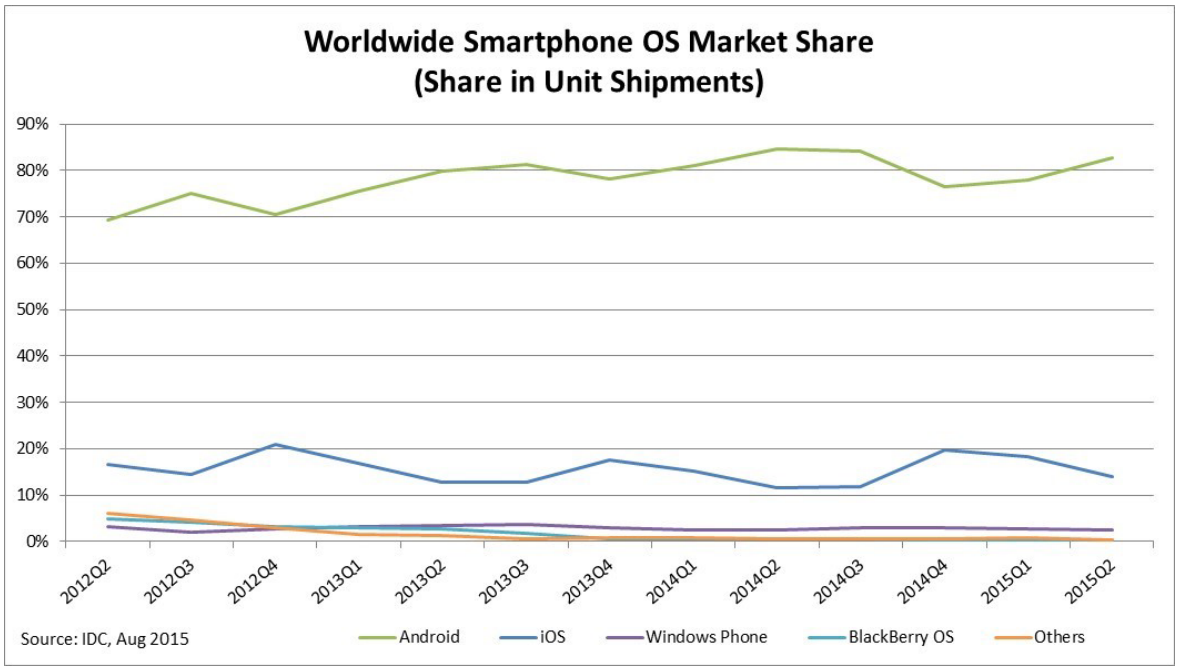

Figura 16 Grafico che mostra l'evoluzione del mercato degli OS mobili [31]

Period	Android	iOS	Windows Phone	BlackBerry OS	Others
2015Q2	82.8%	13.9%	2.6%	0.3%	0.4%
2014Q2	84.8%	11.6%	2.5%	0.5%	0.7%
2013Q2	79.8%	12.9%	3.4%	2.8%	1.2%
2012Q2	69.3%	16.6%	3.1%	4.9%	6.1%

Tabella 1 Market Share del mercato mobile negli ultimi anni [31]

Un altro fattore che ha influenzato questa scelta è la fondamentale importanza che gli smartphone rivestono per l'utente medio.
Una enorme quantità di dati personali è, infatti, detenuta sui telefoni cellulari che diventano vere e proprie banche dati di informazioni sensibili, da account email a codici di carte di credito.
Un meccanismo che permette di trasmettere informazioni, in maniera totalmente celata all'utente, diviene quindi una possibile minaccia per la sicurezza e la privacy dell'utilizzatore.
Una disamina su questo argomento risulta di conseguenza necessaria, per comprendere i pericoli reali e la portata di questo tipo di attacco.

3.1.1. Modello Client/Server

La realizzazione di un canale nascosto avviene seguendo il modello client/server.
Questo modello *indica un'architettura di rete nella quale genericamente un computer client o terminale si connette ad un server per la fruizione di un certo servizio, quale ad esempio la condivisione di una certa risorsa hardware/software con altri client, appoggiandosi alla sottostante architettura protocollare* [32].

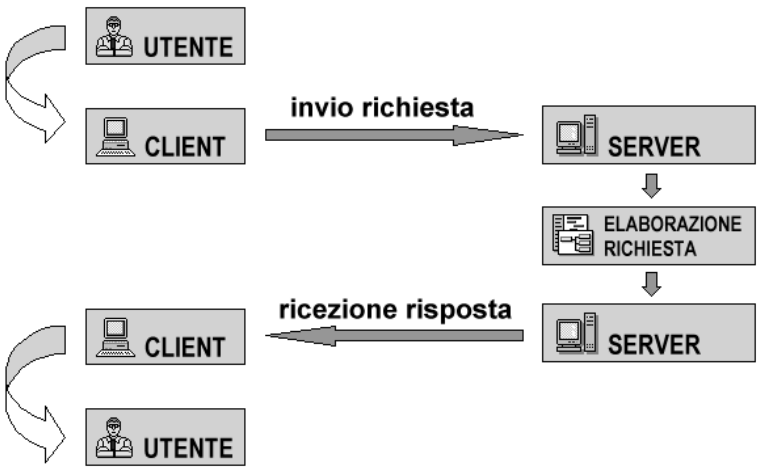

Figura 17 Tipica comunicazione tra un client ed un server [33]

Come riportato nella figura 17, la comunicazione tra un client ed un server è legata all'esaudimento di una richiesta del programma client, di solito caratterizzato da una interfaccia semplice e da un software di limitata complessità i cui compiti sono:

1. Abilitare l'utente a spedire una richiesta di informazione al server;
2. Formattare la richiesta in modo tale che essa sia fruibile al server;
3. Formattare la risposta del server per renderla comprensibile all'utente che sta utilizzando il programma client.

Il server, al contrario, è un programma di elevata complessità con il compito di gestire le diverse richieste dai client che possono giungere nello stesso momento, rendendo quindi necessaria l'implementazione di tecniche per la gestione degli accessi, di allocazione e controllo delle risorse.
Il server si occupa quindi di:

1. Ricevere una richiesta dal client e di processarla;
2. Rispondere, inviando la risposta al mittente della richiesta.

Il modello client/server è particolarmente consigliato per delle reti che necessitano un elevato livello di fiducia, e i suoi principali vantaggi sono:

- *delle risorse centralizzate*: dato che il server è al centro della rete, può gestire delle risorse comuni a tutti gli utenti, come ad esempio un database centralizzato, per evitare i problemi di ripetizione e contraddizione;
- *una sicurezza migliore*: dato il numero di punti d'entrata che permettono l'accesso ai dati è minore;
- *un'amministrazione a livello del server*: considerata la poca importanza dei client in questo modello, hanno meno bisogno di essere amministrati;
- *una rete evolutiva*: grazie a questa architettura è possibile cancellare o aggiungere i client senza disturbare il funzionamento della rete e senza modifiche importanti.

Questa architettura, però, presenta anche degli svantaggi dovuti a:

- *un costo elevato*: dovuto alla tecnicità del server;
- *un anello debole*: il server è il solo anello debole della rete client/server, dato che tutta la rete è strutturata intorno ad esso.

Oltre al modello client/server, per la realizzazione di un canale nascosto sono state utilizzate altre tecnologie che vengono descritte in dettaglio nel paragrafo successivo.

3.1.2. Tecnologie e strumenti utilizzati

Il progetto è stato realizzato adottando il linguaggio Java, standard per le applicazioni Android, e il linguaggio di programmazione C, le cui peculiarità, in termini di prestazioni e di efficienza, sono state sfruttate attraverso l'utilizzo del *Native Development Kit (NDK)*, un toolset distribuito da Google che permette l'integrazione di codice nativo in una applicazione mobile.
Sono stati utilizzati i seguenti strumenti:

- *Android Studio* (versione 1.3.2), IDE per lo sviluppo di applicazioni mobili Android;
- *Native Development kit* (versione r10e).

Android Studio, oramai divenuto IDE standard per lo sviluppo delle applicazioni di questo OS, è stato preferito per le sue molteplici funzionalità uniche [34], riportate nella tabella 2 sottostante che compara i due IDE più utilizzati dagli sviluppatori al momento della stesura di questa tesi, Android Studio appunto, ed Eclipse con l'*Android Development Tool*, delle librerie che vanno integrate a questa piattaforma di sviluppo.

Feature	**Android Studio**	**Eclipse ADT**
Build System	Gradle	Ant
Maven-based build dependencies	Yes	No
Build variants and multiple-APK generation	Yes	No
Advanced Android code completion and refactoring	Yes	No
Graphical layout editor	Yes	Yes
APK signing and keystore management	Yes	Yes
NDK support	Yes	Yes

Tabella 2 Paragone tra le funzionalità offerte dai due IDE [34]

Il *Native Development Kit*, come precedentemente introdotto, permette di integrare del codice nativo, in C o C++, in un'applicazione Android.
Il suo funzionamento viene di seguito illustrato.
Tramite l'utilizzo di uno script chiamato *ndk-build*, il codice nativo viene inviato al "cuore" del NDK che si occupa di selezionare quali file del progetto debbano essere compilati, al fine di produrre codice binario, che viene integrato in quello dell'applicazione che si sta sviluppando.
La comunicazione tra codice Java e nativo avviene attraverso la *Java Native Interface* (JNI), un framework del linguaggio, preposto allo scambio di informazioni tra questi due paradigmi di programmazione.
Riveste una fondamentale importanza uno speciale file chiamato *Android.mk* [35] in cui vengono elencate tutte le risorse (librerie sia native che standard, flags, e file contenenti codice sorgente) che devono essere compilate dal NDK.
Un esempio di Android.mk viene riportato di seguito. Le notazioni utilizzate sono autoesplicative; viene notificato al nucleo del NDK quali sono i documenti da gestire e compilare.

```
include $(CLEAR_VARS)
LOCAL_MODULE := bar
LOCAL_SRC_FILES := bar.c
LOCAL_STATIC_LIBRARIES := foo
include $(BUILD_SHARED_LIBRARY)
```

Figura 18 Un esempio di Android.mk [35]

3.2. Implementazione

L' implementazione di un timing covert channel in ambiente Android è un processo che ha richiesto numerosi passi per il suo completamento.
Durante l'implementazione sono state affrontate diverse sfide, elencate brevemente di seguito, di cui si parlerà nel dettaglio nei paragrafi successivi.
Le problematiche erano legate a:

1. Codifica e decodifica in binario dei messaggi overt e covert;
2. Trasmissione dei singoli bit attraverso lo *User Data Protocol (UDP)*;
3. Sincronizzazione tra client e server;
4. Configurazione ed uso del NDK.

Il primo obiettivo che si è cercato di perseguire è stato quello di implementare un covert channel funzionante in linguaggio C, scelta dettata dalla possibilità di poter gestire in modo appropriato la memoria allocata per l'esecuzione delle operazioni e di conseguenza più efficiente.

3.2.1. Realizzazione in linguaggio C

Il covert channel in C è stato sviluppato come un'applicazione client/server in esecuzione su un sistema Linux Ubuntu 14.04.
La comunicazione tra le due componenti è stata raggiunta attraverso l'utilizzo di socket UDP, il cui *modus operandi* viene di seguito illustrato nella figura 19.
Dopo aver inizializzato le socket la comunicazione avviene tramite i metodi *recvfrom* e *sendto*. Questi due metodi necessitano come parametri l'indirizzo IP e la porta del destinatario, infatti, essendo UDP un protocollo *stateless*, non viene mantenuta traccia delle precedenti interazioni tra client e server.
Lo *User Data Protocol* è un protocollo di tipo *connectionless*, ovvero tra client e server non viene stabilita nessuna sessione e il client non possiede informazioni sull'eventuale arrivo a destinazione di ciò che ha trasmesso.

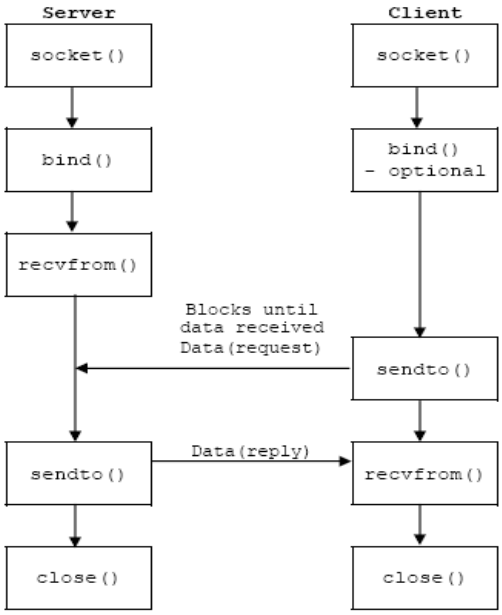

Figura 19 Tipica comunicazione client e server in UDP [36]

La scelta di tale protocollo è stata dettata dalla necessità di inviare pacchetti con contenuti diversi, ma soprattutto per la velocità di trasmissione che permette di raggiungere. UDP, infatti, è il protocollo maggiormente utilizzato per le applicazioni *time sensitive*, ed un covert channel che si basa interamente sul timing esige un protocollo che non perda del tempo nell'invio di *acknowledgment*, procedura standard in TCP.
Inoltre il ruolo di rilievo della temporizzazione nella trasmissione richiede un invio di datagrammi singoli (come descritto nello schema ON/OFF del capitolo 2), servizio offerto da UDP, e non come uno *stream* di dati come realizzato da TCP.
Viene di seguito illustrato nel dettaglio il funzionamento del client e del server, descrivendone gli algoritmi utilizzati ed elencando le librerie impiegate.

3.2.1.1. Client

Lato client sono state impiegate varie librerie le cui funzioni vengono riportate nella tabella che segue.

Libreria	**Descrizione**
`stdio.h`	Gestione I/O
`stdlib.h`	Gestione della memoria
`sys/types.h`	Librerie per la gestione delle socket in C, definiscono alcune delle strutture necessarie per l'inizializzazione delle socket.
`sys/socket.h`	
`netinet/in.h`	
`netdb.h`	
`time.h`	Funzioni per la gestione del tempo
`string.h`	Funzioni di manipolazione delle stringhe

Tabella 3 Librerie impiegate dal Client in C

L’importanza del client risiede soprattutto in due mansioni particolari che esso deve svolgere, ovvero:

1. Codifica in binario dei messaggi overt e covert;
2. Invio dei singoli bit secondo lo schema on/off.

Dopo aver ricevuto in input (da file o da tastiera) i due tipi di messaggi da trasmettere, il client invoca la funzione *int* encode (char* message)*, il cui corpo viene di seguito riportato e commentato.

```
int* encodeChar (char c) {
      int *output=(int*)calloc(8,sizeof(int));
      int i;
      for (i=0;i<8;i++) {
            output[7-i] = ((c>>i)& 1);
      }
      return output;
}
```

Tabella 4 Il metodo che si occupa della codifica in binario di un carattere

```
int* encode(char* message) {
      int length=(int)strlen(message);
      int* encoded = (int*)calloc(8*length,sizeof(int));
      int i,j;
      int pos=0;
      int* single_char=(int*)calloc(8,sizeof(int));
      for (i=0;i<length;i++) {
            single_char=encodeChar(message[i]);
            for(j=0;j<8;j++)
                  encoded[j+pos]=single_char[j];
            pos= pos+8;
        }
      return encoded;
}
```

Tabella 5 metodo che codifica una stringa in binario

La funzione *encode* codifica una stringa passata come parametro con una rappresentazione a 8 bit per carattere. La funzione inizializza in maniera dinamica due aree di memoria attraverso la chiamata alla funzione *calloc*, presente nella libreria *stdlib.h*, una per il messaggio codificato finale ed una per il singolo carattere, la cui codifica viene

commissionata alla funzione *int* encodeChar (char c).* Quest'ultima effettua in loop uno shift e un *AND* logico per estrarre un singolo bit dal carattere e porlo in un array di interi.
La funzione *encode* si occupa poi di inserire correttamente le sequenze di 8 bit per carattere che *encodeChar* produce, restituendo alla funzione che l'ha invocata un array contenente la stringa codificata in binario.
Terminata la codifica in binario delle stringhe overt e covert, il client deve trasmettere queste sequenze di bit secondo i dettami dello schema on/off. Questo risultato viene ottenuto utilizzando l'algoritmo in figura.

```
for (i=0;i<strlen(message)*8;i++) {
     if (i==strlen(message)*8-1 && z<length_covert)
         i=0;
      if(z<length_covert) {
         if (encoded_covert[z]==1) {
            usleep(timing_interval/2);
            sendto(sock,  &encoded_overt[i],sizeof(encoded_overt[i]),  0,
(struct sockaddr *)&addr,sizeof(addr));
            usleep(timing_interval/2);
            }
            else if (encoded_covert[z]==0) {
                  usleep(timing_interval);
                  i--;
            }
            z++;
      }
      else
          sendto(sock,   &encoded_overt[i],sizeof(encoded_overt[i]),   0,
(struct sockaddr *)&addr, sizeof(addr));

}
```

Tabella 6 trasmissione implementata secondo l'on/off scheme

L'algoritmo viene eseguito un numero di volte pari alla lunghezza del messaggio covert codificato in binario, infatti, nel caso in cui il messaggio "illecito" sia più lungo del messaggio "lecito" il client ritrasmette nuovamente il messaggio overt fino al completo invio dei bit covert.
L'invio dei datagrammi legittimi avviene secondo lo schema on/off, con l'invio di un pacchetto nell'intervallo di tempo concordato per comunicare un bit 1 covert e con un periodo di silenzio per inviare un bit 0 illegittimo.
Questo periodo di pausa viene implementato tramite la funzione *usleep (**useconds_t** usec)* che pone in pausa il thread corrente per il numero di microsecondi specificati.

3.2.1.2. Server

Il server opera anch'esso due funzioni molto importanti e complementari a quelle del client, ovvero:

1. Interpretazione dello schema on/off;
2. Decodifica dei dati binari ricevuti.

Per implementare queste funzioni sono state impiegate le stesse librerie utilizzate dal client, per le specifiche, quindi, si rimanda al paragrafo precedente.
La prima operazione svolta dal receiver è quella di inizializzare la socket e porsi in ascolto di eventuali trasmissioni da parte dei client.
Dopo aver inizializzato i parametri della socket (come la porta di ascolto, il tipo di socket e i permessi di ricezione dei pacchetti), viene invocata la funzione *bind* che associa la socket all'indirizzo IP e alla porta specificati.

```
if ( (sock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("Socket creation error");
    exit(-1);
  }

  memset((void *)&addr, 0, sizeof(addr));
  addr.sin_family = PF_INET;
  addr.sin_port = htons(1745);
  addr.sin_addr.s_addr = htonl(INADDR_ANY);
  len=sizeof(addr);
  if (bind(sock, (struct sockaddr *)&addr,sizeof(addr)) < 0) {
    perror("bind error");
    exit(-1);
  }
```

Tabella 7 Creazione di una socket per la comunicazione

Dopo aver stabilito la comunicazione, il server si pone in attesa dei datagrammi in uscita dal client, interpretando l'*interpacket gap* tra un pacchetto e il successivo secondo l'algoritmo mostrato in figura.

```
 clock_gettime(CLOCK_MONOTONIC, &now);
 n=recvfrom(sock,&bitOvert,sizeof(int),0,(struct sockaddr*) &addr,  &len);
 clock_gettime(CLOCK_MONOTONIC, &after);
 difference=((double)after.tv_sec + 1.0e-9*after.tv_nsec)-
((double)now.tv_sec + 1.0e-9*now.tv_nsec);
 difference=difference-timing_interval/2;
 int zeros=(int)(difference/timing_interval);
 if(difference>0)
    insertCovert(&covert,zeros,&length_covert);
```

Tabella 8 implementazione della decodifica degli intervalli temporali osservati dal server

L'algoritmo sfrutta una caratteristica del metodo *recvfrom* delle socket, ovvero questa funzione, una volta invocata, si pone in uno stato di attesa fino a che un pacchetto non giunge al server.

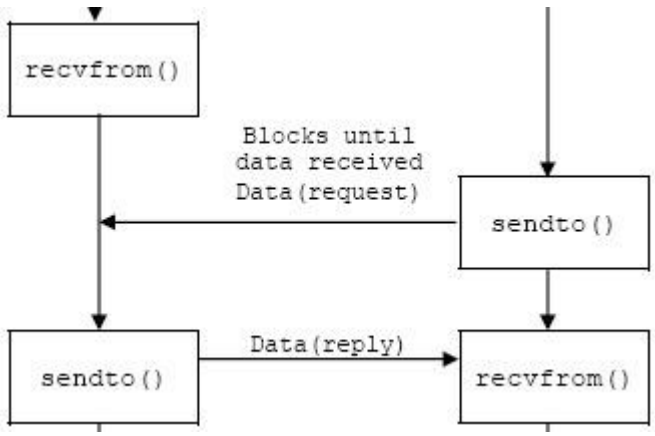

Figura 20 La funzione recvfrom si pone in attesa fino all'arrivo di un datagramma [36]

Questo tempo di "pausa" viene misurato attraverso le chiamate alla funzione *clock_gettime* subito prima e dopo l'esecuzione della funzione *recvfrom*, salvando il tempo restituito dal metodo in due variabili e calcolandone la differenza.
Questo periodo di sospensione viene interpretato dal server secondo la seguente procedura:

1. Viene sottratto metà dell'intervallo concordato al periodo di pausa calcolato, infatti, la funzione *recvfrom* interrompe il suo funzionamento fino all'arrivo di un datagramma, che viene inviato a metà intervallo, quando si incontra un bit covert pari ad 1;
2. Il risultato viene diviso per il time slot, ottenendo un numero intero che indica il numero di bit di valore 0 che precedono il bit covert di valore 1;
3. Vengono inseriti questi valori in una lista dinamica tramite la funzione *insertCovert.*

L'istruzione di controllo *if (difference>0)* è necessaria per verificare che i pacchetti giunti al server siano stati comunque inviati secondo lo schema on/off e di conseguenza ci siano ancora dei bit del messaggio covert da dover interpretare.

Terminata la ricezione dei datagrammi, il server invoca la funzione *decode*, il cui corpo viene riportato di seguito, che ha il compito di trasformare l'array di valori binari in una stringa.
La funzione *decode* svolge un lavoro inverso rispetto alla funzione *encode* del client precedentemente analizzata. Seguendo lo schema di una codifica a 8 bit per carattere, la funzione itera sulla sequenza binaria passata come parametro, processando sequenze di 8 bit alla volta.
La decodifica viene conseguita attraverso la funzione *strtoul* che prende in input queste sequenze di 8 valori binari trasformandoli nel carattere corrispondente.
I caratteri processati poi vengono memorizzati in un array che viene restituito come risultato della funzione.

```
char* decode(char* x) {
  int len=strlen(x)/8;
  int i,j;
  int pos=0;
  char temp[9];
  char c;
  char* endP=NULL;
  char* decoded=(char*)calloc(len+1,sizeof(char));
  for(i=0; i<len; i++) {
    for (j=0;j<8;j++) {
      temp[j]=x[j+pos];
    }
    c=strtoul(temp,&endP,2);
    decoded[i]=c;
    pos=pos+8;
  }
  decoded[len]='\0';
  return decoded;
}
```

Tabella 9 Metodo che si occupa della decodifica del messaggio covert

Il covert channel così implementato ha dato risultati soddisfacenti, fornendo quindi una base dalla quale è stato sviluppato il progetto in Android. Nel capitolo 4 verranno poi comparate le prestazioni di questa realizzazione con quella in Android, mettendo in risalto punti di forza e debolezza di entrambe le implementazioni.

3.2.2. Struttura dell'applicazione Android

Prima di analizzare in dettaglio l'organizzazione e il codice dell'applicazione sviluppata è necessario descrivere alcune procedure preliminari che sono necessarie per il corretto funzionamento del programma.

Le applicazioni Android, di default e per motivi di sicurezza, non offrono libero accesso a internet ad una applicazione.
Risulta quindi necessario aggiungere i seguenti permessi nel manifest dell'applicazione, denominato appunto *AndroidManifest.xml.*

```
<uses-permission android:name="android.permission.INTERNET" />
```

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Tabella 10 permessi necessari per l'accesso a internet dell'applicazione Android

L'*AndroidManifest.xml comunica informazioni fondamentali sulla tua applicazione al sistema Android, informazioni che il sistema deve avere prima di poter eseguire qualsiasi parte di codice* così come riportato sulla guida ufficiale di Android [37].
Un'altra operazione preliminare è la configurazione del NDK in Android Studio, per permettere quindi al progetto di poter compilare ed eseguire il codice nativo realizzato in C/C++.
Questa procedura, al momento della stesura della tesi, è in fase beta in Android Studio (versione 1.3.2), fattore che l'ha resa piuttosto ostica.
Per utilizzare in modo corretto queste funzionalità è necessario prima di tutto scaricare la versione r10e del NDK, l'unica supportata in questo momento e, dopo averla resa disponibile all'IDE, impostare come Gradle (software per sviluppi multi-progetto, utilizzato per la configurazione dello stesso) la nuova versione sperimentale 0.2.0.
Sono necessarie anche delle aggiunte ad altri file di configurazione che qui non vengono riportate per motivi di semplicità, i dettagli sono però disponibili nella seguente guida: http://tools.android.com/tech-docs/new-build-system/gradle-experimental.
L'applicazione Android sviluppata si compone da due activity:

1. MainActivity;
2. SendActivity.

3.2.2.1. MainActivity

Questa activity ha il compito di inizializzare la connessione con il server (precedentemente avviato e in ascolto su una porta predefinita) e gestire tutte le problematiche (server non raggiungibile, porta occupata) che si possono verificare.
Per implementare queste funzionalità sono state impiegate varie librerie, le cui descrizioni sono riportate nella tabella 3 di seguito.

Librerie	Descrizione
java.net.InetAddress	Funzioni per la gestione degli indirizzi IP
java.net.DatagramPacket	Include le funzioni per l'invio di datagrammi Udp
java.net.DatagramSocket	Funzioni per socket Udp
java.io.IOException	Eccezioni di I/O e di host sconosciuto
java.net.UnknownHostException	
android.os.AsyncTask	Permette l'utilizzo della classe AsyncTask e delle sue funzionalità

Tabella 11 Librerie utilizzate per lo stabilimento della connessione tramite socket

L'activity presenta una interfaccia minimale in cui vengono richiesti all'utente i parametri della connessione con il receiver, quali il suo indirizzo IP e la porta sulla quale il server è in ascolto.
La connessione con il server viene stabilita tramite l'impiego di una sottoclasse di tipo AsyncTask. Questa classe è stata scelta per ottimizzare l'utilizzo dell'interfaccia utente, rendendola indipendente dalle procedure di consolidamento della connessione che vengono eseguite in background [38]. Ottenuti i parametri dall'interfaccia utente, la classe si occupa di creare una socket di tipo UDP e di inviare un datagramma speciale in cui viene comunicato al server della presenza di un client che vuole iniziare una trasmissione.

```
socket = new DatagramSocket();
String sof="Start of Frame";
byte[] sendData=sof.getBytes("UTF-8");
InetAddress ipAddress=InetAddress.getByName(dstAddress);
DatagramPacket startOfFrame=new
DatagramPacket(sendData,sendData.length,ipAddress,dstPort);
socket.send(startOfFrame);
while(!received) {
      DatagramPacket receivePacket = new DatagramPacket(sendData,
sendData.length);
      socket.receive(receivePacket);
      String acknowledgement = new String(receivePacket.getData(), 0,
receivePacket.getLength(), "UTF-8");
      if (acknowledgement.equals("SOF received")) {
          response = "Connection Established";
          received=true;
          }
}
```

Tabella 12 metodo che stabilisce la comunicazione tra client e server

Se il client riceve una risposta positiva dal server allora viene notificato all'utente che la connessione è stata stabilita e viene lanciata la SendActivity che si occuperà della trasmissione vera e propria.

```
if(response.equals("Connection Established")) {
   Intent = new Intent(MainActivity.this,
SendActivity.class);
   intent.putExtra("dstAddress",
editTextAddress.getText().toString());
   intent.putExtra("dstPort",
Integer.parseInt(editTextPort.getText().toString()));
          startActivity(intent);
}
```

Tabella 13 metodo che lancia la SendActivity una volta ricevuta una risposta positiva dal server

I principali metodi di questa Activity vengono riportati nella tabella 14 che segue, con una breve descrizione delle loro funzionalità principali.

Metodo	Funzione
`protected void onCreate(Bundle savedInstanceState)`	Si occupa della inizializzazione della interfaccia utente, gestendo i parametri provenienti da input
`public void onClick(View arg0)`	Listener per i bottoni dell'interfaccia utente, viene attivato alla pressione di un bottone
`protected Void doInBackground(Void... arg0)`	Metodo della sottoclasse AsyncTask, stabilisce la connessione in background rispetto alla UI
`protected void onPostExecute(Void result)`	Aggiorna la UI con i risultati ottenuti dal lavoro in background del metodo precendente

Tabella 14 Listato dei metodi principali della MainActivity

3.2.2.2. SendActivity

Questa classe ha il compito di gestire la trasmissione dei messaggi overt e covert dal client al server e di conseguenza della instaurazione vera e propria di un canale nascosto.
Le librerie utilizzate da questa Activity sono tutte librerie standard, che non fanno altro che includere le componenti fondamentali di Android, come i Widget e le View e pertanto non vengono qui riportate.
L'activity ha il compito di gestire la trasmissione dei messaggi overt e covert dal client al server e di conseguenza della instaurazione vera e propria di un canale nascosto.
La prima funzione offerta da questa classe è una semplice interfaccia in cui è possibile settare i messaggi overt e covert che si vogliono trasmettere e con quale temporizzazione.
La comunicazione, invece, avviene tramite le funzionalità offerte dal *native development kit*, infatti, questa activity utilizza una versione modificata della funzione di invio analizzata nel covert channel realizzata in C.

L’utilizzo del codice nativo avviene attraverso delle funzioni che vengono di seguito analizzate in dettaglio.
Il primo metodo riportato è quello che permette al sistema di caricare le librerie e i file definiti nel file *Android.mk.* Senza questa funzione nulla sarebbe utilizzabile, è necessario quindi configurare in maniera ottimale questi file prima di procedere.

```
static {
        System.loadLibrary("hello-jni");
    }
```

Tabella 15 Metodo che permette il caricamento delle librerie in C nel codice Java

Una volta che le librerie sono state correttamente caricate all’interno del sistema, la SendActivity chiama un metodo JNI di cui conosce solo la firma ma non il corpo, presente nel file di codice sorgente C caricato precedentemente.
Il metodo sopra citato è il seguente.

```
public native int sendFromJNI(String address, int port, String
overt,String covert,int interval);
```

Tabella 16 Metodo JNI che fa da tramite tra linguaggio Java e C

Come specificato nel framework JNI, i metodi che utilizzano codice nativo devono essere identificati tramite la parola chiave *native* ed il corpo del metodo, sotto riportato, permette il passaggio da codice Java a codice nativo.
Questa funzione ha il compito di trasformare i parametri provenienti dal codice java a parametri fruibili al codice in C, convertendo quindi gli oggetti in puntatori ad aree di memoria, e di invocare la funzione in C che si occupa della trasmissione al server.
Come già precedentemente citato, la funzione in C è molto simile a quella già analizzata nel covert channel completamente in linguaggio nativo, in questa versione, però, viene introdotto un pattern di sincronizzazione che sarà discusso con più attenzione nel paragrafo successivo.

```
jint Java_com_example_client_SendActivity_sendFromJNI( JNIEnv* env,
jobject this ,jstring address, jint port,jstring overt,jstring covert,jint
interval) {
    const char *nativeAddress = (*env)->GetStringUTFChars(env, address,
0);
    const char *nativeOvert = (*env)->GetStringUTFChars(env, overt, 0);
    const char *nativeCovert = (*env)->GetStringUTFChars(env, covert, 0);
    int nativePort=(int)port;
    int nativeInterval=(int)interval;
    jint result=
createAndSendSocket(nativeAddress,nativePort,nativeOvert,nativeCovert,nati
veInterval);
    return result;
}
```

Tabella 17 Corpo del metodo JNI

La firma e le funzionalità dei metodi principali di questa Activity viene quindi riportato nella tabella seguente.

Metodo	Funzione
`protected void onCreate(Bundle savedInstanceState)`	Inizializza l'interfaccia dell'activity e gestisce i dati provenienti da input
`public native int sendFromJNI(String address, int port, String overt,String covert,int interval);`	metodo JNI che invoca il metodo in C
`public void sendMessage()`	Invoca il metodo JNI alla pressione del pulsante send della UI

Tabella 18 Listato dei metodi della SendActivity

3.3. Sincronizzazione

La sincronizzazione è una delle problematiche, e allo stesso tempo delle sfide, più grandi che devono essere affrontate nella realizzazione di un timing covert channel, specialmente in ambito mobile. Come ampiamente discusso nel capitolo 2, il timing covert channel "affida" la sua capacità di trasmettere un messaggio in maniera nascosta alla manipolazione volontaria dell'*interpacket gap*, ovvero il tempo che intercorre tra l'invio di un datagramma e il suo successivo.
Il client e il server operano spesso con clock diversi e considerando i ritardi inevitabili inseriti dal canale, raggiungere una sincronizzazione perfetta è pressoché impossibile.
Il *time slot*, concordato a priori da client e server, influenza in modo significativo le prestazioni del canale nascosto. Valori relativamente bassi per questo intervallo incrementano

di molto la possibilità di una perdita di sincronizzazione; valori più alti, al contrario, tendono a “compensare” l’effetto dei ritardi introdotti dalla rete a scapito di una riduzione della banda del canale.
Si rendono quindi indispensabili degli algoritmi il cui scopo sia quello di rendere meno drammatici gli effetti sopra citati mantenendo un buon livello di simultaneità degli eventi. Viene di seguito esplicato l’algoritmo maggiormente utilizzato nella implementazione del covert channel presentata.

3.3.1. Start of Frame delimiter

La metodologia che è stata impiegata per mantenere alto il livello di sincronizzazione (e di conseguenza rendere minimo l’effetto del jitter del network) tra client e server si basa sull’invio periodico di una speciale sequenza, nota a priori da entrambi gli attori presenti in scena.
Nella figura che segue viene riportato il codice utilizzato per l’invio della sequenza di sincronizzazione da parte del client al server.
Il pattern di sincronizzazione viene inviato all’inizio della trasmissione o dopo l’invio completo di un carattere covert come evidenziato dalla condizione
`if ( start==1 || cont==8).`
Questo pattern, inviato anch’esso secondo lo schema ON/OFF, permette al server di allineare il proprio clock con quello del client. Infatti, notifica al receiver che una nuova sequenza di bit covert (un carattere quindi del messaggio celato che si vuole inviare) è in procinto di essere trasmessa, permettendo quindi la compensazione dei ritardi introdotti dal canale ed una migliore decodifica lato server.

```
   if ( start==1 || cont==8) {
       for ( z=0; z<8;z++) {
           if (sincro[z]==1) {
               usleep((useconds_t) (timing_interval/2));
               sendto(sock,         &encodedOvert[i],sizeof(int),0,(struct
sockaddr *)&addr,sizeof(addr));
               usleep((useconds_t) (timing_interval/2));
               i++;
           }
           else if (sincro[z]=
                    usleep((useconds_t) timing_interval);
        }
        cont=0;
        start=0;
   }
```

Tabella 19 Metodo per l'invio del pattern di sincronizzazione

La decodifica avviene secondo il seguente algoritmo.

```
zeros=(int)(diff/interval);
for ( z=0; z<zeros;z++)
    single_char.add(0);
single_char.add(1);

if (single_char.size()>=8) {
    List<Integer> character=single_char.subList(0,8);
    List<Integer> overflow=new LinkedList<>();
    if(single_char.size()>8)
       overflow=single_char.subList(8,single_char.size());
    syn=decode(character);
    if (syn.equals("}"))
        syncroReceived=true;
}
```

Tabella 20 Metodo per la decodifica lato server

Il server memorizza i bit covert, interpretati dal ritardo con cui gli giungono i pacchetti leciti, in una lista che, una volta contenente i valori di 8 bit covert, viene decodificata.
Se il carattere decodificato corrisponde a quello di sincronizzazione, il server notifica a sé stesso che un nuovo carattere nascosto è in procinto di essere trasmesso dal client e si prepara a compensare i ritardi introdotti dal canale secondo lo schema di seguito riportato.

```
if (syncroReceived && syncro<8) {
    if (seconds<interval)
        seconds=interval;
    else {
          double result=seconds/interval;
          int loss=(int)Math.round(result);
          seconds=loss*interval;
       }
     syncro++;
}
else {
      syncro=0;
      syncroReceived=false;
      }
```

Tabella 21 Metodo che aggiusta il sincronismo dei pacchetti arrivati

Il receiver analizza l'*interpacket gap* osservato tra un pacchetto ed il suo predecessore, arrotondando questa differenza di tempo al valore del *time slot* (o di un suo multiplo) più vicino, rendendo quindi la decodifica più affidabile. Questo procedimento viene ripetuto fino all'interpretazione di 8 bit nascosti, dopo di che il meccanismo di ricezione torna ad essere quello standard.

Al contrario, nel caso in cui ad essere giunto al server sia un normale carattere covert, la sequenza decodificata viene inserita nella lista che conterrà il messaggio nascosto trasmesso dal sender.
L'analisi delle prestazioni dell'applicazione con e senza pattern di sincronizzazione viene lasciata al capitolo 4, nel quale sarà presente una indagine più approfondita di queste tematiche.

4 Analisi delle prestazioni

In questo capitolo vengono esaminate le performance dell'applicazione realizzata, mettendo in risalto le analogie e le differenze tra il covert channel sviluppato in linguaggio C e lo stesso in Android.
Come già discusso nel capitolo 3, i fattori che influenzano in maniera critica l'accuratezza di un canale nascosto sono lo stato della rete al momento della trasmissione e il timing slot con cui vengono inviati i pacchetti leciti.
Durante i test che sono stati effettuati è stato variato l'intervallo di tempo tra un datagramma e il successivo, in un range che oscilla dai 10 ai 100 millisecondi, in modo da poter osservare l'alterazione delle performance correlata a questo fattore data l'impossibilità di poter manipolare a proprio piacimento lo stato del network.

4.1. Dispositivi utilizzati

Al fine di garantire una maggiore precisione nella definizione dei test, l'applicazione è stata testata utilizzando i seguenti dispositivi:

- Asus N550JV:
  - Hardware:
    - Processore: Intel Core i7-4700HQ 2.40 GHz;
    - Grafica: Intel Hashwell mobile;
    - Memoria installata (RAM): 16 Gb;
  - Software:
    - Linux Ubuntu 14.04 LTS
- Samsung Galaxy S Advance I9070:
  - Hardware:
    - Processore: Dual-core 1 GHz Cortex-A9;
    - Memoria installata (RAM): 768 Mb;
  - Software:
    - Android OS Gingerbread (versione 2.36)
- iMac:
  - Hardware:
    - Processore: Intel Core i3 3,2 GHz
    - Grafica: ATI Radeon HD 5670 512 Mb
    - Memoria installata (RAM): 12 Gb
  - Software:
    - OS X Yosemite (Versione 10.10.3)

Lo smartphone impiegato è un cellulare di media fascia, *target* ideale per i test a causa della sua vasta diffusione sul mercato con la conseguente accessibilità ad un alto numero di utenti.

I dati relativi agli esperimenti compiuti vengono descritti nel dettaglio nel prossimo paragrafo.

4.2. Prestazioni in Local host

Lo stretto legame tra un covert channel e lo stato della rete rende i test effettuati su una rete locale non particolarmente indicativi, a causa dell'impossibilità di osservare il mutamento delle prestazioni del covert channel dovuto al rumore e ai ritardi introdotti dal canale.
L'utilità di questa sperimentazione risiede, però, nella verifica della correttezza dell'algoritmo, mostrandone il completo funzionamento in un caso ideale di trasmissione che, essendo tale, non può però trovare riscontro nella realtà.

4.2.1. Prestazioni del Covert channel realizzato in Android

A causa di un problema riscontrato con l'emulatore Android, non è stato possibile testare l'applicazione in locale. Il suddetto problema, scoperto durante lo sviluppo, è causato da una introduzione di ritardi (di natura software e indipendenti dalla realizzazione dell'applicazione stessa) consistenti da parte dell'emulatore, che inficiano la sincronizzazione tra client e server, rendendo, di fatto, poco affidabili le statistiche raccolte durante questi test.
Al contrario, sono stati svolti diversi test sul covert channel implementato in C, che hanno messo in risalto diversi aspetti.

4.2.2. Prestazioni del Covert Channel realizzato in C

I test effettuati sull'implementazione del canale nascosto in linguaggio C hanno restituito dei risulti ottimi per quanto riguarda l'affidabilità di questo tipo di trasmissione.
Infatti, come già introdotto, essendo una rete locale, non si sono presentati i fenomeni di aumento della latenza della rete e di un uso dello stesso da parte di più host, fattori che hanno permesso una trasmissione corretta del messaggio celato nel 100% dei casi.
Gli esperimenti che sono stati condotti sono stati molteplici e l'intervallo tra un pacchetto e il successivo è stato variato in un range che va dai 5 ai 100 millisecondi, osservando nella totalità dei casi l'invio corretto di tutti i bit covert con la conseguente decodifica esatta del messaggio nascosto.
La lunghezza di quest'ultimo è stata anch'essa modificata nel corso delle prove, passando da messaggi relativamente brevi ad altri di lunghezze superiori ai 5000 caratteri.
I dati non vengono qui riportati per il loro scarso interesse.

4.3. Prestazioni del covert channel tra due dispositivi fissi

Il canale nascosto è stato testato attraverso la comunicazione client/server tra due computer, le cui specifiche sono riportate nel paragrafo 4.1.
La trasmissione è avvenuta tramite una rete Wi-Fi a 3 Mbps (stima calcolata attraverso lo *speed test* offerto da *Ookla* [39]).
Le prestazioni dell'invio dell'informazione celata sono state osservate al variare del time slot, della lunghezza del messaggio e naturalmente dello stato della rete.
Durante i test, l'intervallo di tempo tra due datagrammi ha assunto valori compresi tra 5 e 100 millisecondi.
La stringa utilizzata per la sperimentazione conteneva un indirizzo email fittizio, una classica tipologia di informazione che potrebbe essere di alto interesse per un hacker.
I test hanno mostrato come il messaggio "mario.rossi@gmail.com" venga inviato correttamente nella totalità dei casi per time slot ampi, ovvero che variano tra i 25 e i 100 millisecondi, i cui dati, poco indicativi, non vengono riportati.
Di maggiore interesse, invece, sono i dati, riportati nelle tabelle 22 e 23, riguardanti l'invio del messaggio con time slot di 5 e 7 millisecondi.

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	98	7	328
2	100	0	328
3	90	32	328
4	100	0	328
5	99	2	328
6	100	0	328
7	99	3	328
8	100	0	328
9	100	0	328
10	100	0	328

Tabella 22 Invio del messaggio con un time slot di 7 ms

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	85	48	328
2	73	88	328
3	90	32	328
4	88	39	328
5	70	98	328
6	75	82	328
7	88	37	328
8	96	13	328
9	81	63	328
10	95	16	328

Tabella 23 Invio della stringa a 5 ms

Come indicato dai dati in Tabella 22, la trasmissione con un time slot di 7 millisecondi offre una comunicazione piuttosto affidabile, con la rilevazione di errori in un numero limitato di casi e su una bassa percentuale di bit del messaggio covert. Il numero piuttosto elevato di

errori nel tentativo numero 3 può trovare una sua spiegazione in un aumento della latenza introdotta dalla rete, con la conseguente perdita di sincronizzazione tra sender e receiver.
I dati evidenziano il calo netto delle prestazioni all'abbassamento del time slot a 5 millisecondi. Sotto queste condizioni, infatti, la trasmissione diviene particolarmente problematica con una elevata quantità di bit decodificati erroneamente da parte del receiver che non è in grado, quindi, di decifrare e di usufruire dell'informazione illecita inviata dal client.

4.4. Prestazioni del covert channel implementato su Android

L'applicazione Android è stata testata tramite una comunicazione client/server instaurata tra lo smartphone e il computer, per le cui specifiche si rimanda al paragrafo 4.1, tramite l'utilizzo di una rete WI-FI con una velocità di 3Mbps (stima ottenuta tramite il test di velocità di *Ookla* [39]), al momento della sperimentazione.
Il testing del canale nascosto ha messo in risalto una differenza notevole nelle performance in base alla modifica di 3 variabili che sono: il timing slot, la tipologia di messaggio nascosto che si vuole inviare e lo stato della rete.
Per le stesse ragioni spiegate nel precedente paragrafo, il primo messaggio impiegato per i test consisteva in una stringa contenente un indirizzo email.
Il messaggio "mario.rossi@gmail.com", utilizzato per la sperimentazione, è stato inviato con diversi timing slot che hanno condizionato molto la resa effettiva del covert channel e la conseguente riuscita della trasmissione.
In Tabella 24 vengono riportati i dati relativi alla spedizione della suddetta stringa con un timing slot di 100 millisecondi, valore piuttosto alto, ma che consente di osservare la variazione delle performance innescate soprattutto dallo stato della rete.

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	85	24	328
2	100	0	328
3	89	18	328
4	100	0	328
5	99	2	328
6	100	0	328
7	99	1	328
8	100	0	328
9	100	0	328
10	100	0	328

Tabella 24 Trasmissione del messaggio "mario.rossi@gmail.com" con un intervallo di 100 ms

Come evidente dai dati, tranne in due casi, ovvero i tentativi 1 e 3, dove si è verificato un rallentamento evidente della rete, il canale nascosto riesce ad trasmettere correttamente l'informazione sotto queste condizioni.

L’ampio intervallo tra i datagrammi limita l’effetto dei ritardi che il canale può, eventualmente, introdurre, consentendo quindi al server di osservare variazioni di tempo abbastanza regolari e di conseguenza di decodificare correttamente il messaggio.
Gli errori osservati negli esperimenti sono, principalmente, di due tipologie:

1. Errore di shift: il server riceve un datagramma dopo un intervallo di tempo che può essere interpretato in maniera controversa, causando una decodifica errata con la conseguente aggiunta di bit covert;
2. Errore di *bit flip*: ovvero su un singolo bit, decodificando uno 0 al posto di un 1 o viceversa.

Gli errori della prima categoria sono i più comuni e allo stesso tempo i più dannosi per i canali nascosti.
Infatti, come dimostrato dalla prova sperimentale, un solo shift può compromettere la decodifica di gran parte del messaggio celato. Nel tentativo 7, pur avendo inviato correttamente il 99% dei bit, i caratteri che componevano il suffisso “com” della stringa nascosta erano stati rimpiazzati da simboli senza significato.
Nel capitolo 3 è stata illustrata la necessità di un pattern di sincronizzazione che limiti gli effetti di questi errori, tuttavia, l’imprecisione può riscontrarsi proprio durante l’invio di questi bit, come in questo caso, rendendo la problematica impossibile da risolvere del tutto.
Al contrario, la seconda tipologia di errori può causare la modifica di un carattere, in maniera lieve se ad essere decodificato male siano uno o due bit, trasformando, nella maggior parte dei casi, un carattere nel suo successivo nella tabella ASCII.
Abbassando il timing slot a 75 millisecondi, si ottengono comunque risultati ottimi per quello che riguarda l’invio e la decodifica del messaggio così come riportato in tabella 23.

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	100	0	328
2	70	49	328
3	100	0	328
4	98	2	328
5	100	0	328
6	95	8	328
7	100	0	328
8	43	95	328
9	100	0	328
10	99	1	328

Tabella 25 Invio del messaggio “mario.rossi@gmail.com” con un timing slot di 75 ms

I dati riportati in tabella sono piuttosto indicativi, ma è importante sottolineare come la manifestazione di un errore sui bit delle prime sequenze di sincronizzazione possa inficiare completamente la comunicazione. Infatti, un errore di decodifica di questi dati non permette al server di allineare il proprio clock con quello del client, accumulando un ritardo o un anticipo che influenza la decodifica di tutti i bit successivi.

Questo spiega l'alto numero di errori che si osserva nel tentativo numero 8, che potrebbe essere causato da un solo bit errato in una delle prime sequenze di sincronizzazione.
Nelle tabelle 26 e 27 vengono mostrati i dati relativi all'invio del messaggio abbassando ulteriormente il time slot, fissandolo prima a 50 e poi a 25 millisecondi.
Come si può osservare, il canale nascosto comincia a perdere efficacia nel secondo caso, fattore dovuto alla perdita della capacità di compensazione dei ritardi dall'intervallo di tempo.
I ritardi che incorrono durante la trasmissione, infatti, si avvicinano come valore a quello dell'*interpacket gap* influenzando la decodifica del server.

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	47	89	328
2	100	0	328
3	88	20	328
4	100	0	328
5	100	0	328
6	40	100	328
7	86	22	328
8	100	0	328
9	100	0	328
10	61	65	328

Tabella 26 Invio del messaggio "mario.rossi@gmail.com" a 50 ms

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	47	88	328
2	64	60	328
3	43	95	328
4	56	73	328
5	55	75	328
6	38	104	328
7	48	86	328
8	72	47	328
9	53	78	328
10	57	72	328

Tabella 27 Invio del messaggio "mario.rossi@gmail.com" a 25 ms

La situazione peggiora ulteriormente se si porta il time slot a 10 millisecondi, ottenendo una trasmissione errata di molti bit nella totalità dei casi come mostrato in Tabella 28.
La differenza di prestazioni tra la versione del canale nascosto implementato in C e quello in Android è notevole.
Infatti, come già analizzato nel paragrafo 4.3, la realizzazione del canale celato in un linguaggio di basso livello come il C ha portato ad una trasmissione corretta anche per intervalli di tempo prossimi ai 5-7 millisecondi.

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	52	80	328
2	41	98	328
3	41	98	328
4	37	105	328
5	63	61	328
6	42	97	328
7	71	48	328
8	68	53	328
9	64	59	328
10	67	55	328

Tabella 28 Dati dell'invio del messaggio "mario.rossi@gmail.com" a 10 ms

Questa circostanza ha sollevato diversi dubbi e questioni durante la fase di realizzazione, soprattutto dovute alla implementazione molto simile tra i client nei due casi, considerando anche che il client Android richiama lo stesso metodo utilizzato dal sender in C, rendendo, quindi, piuttosto singolare questa differenza tra l'una e l'altra realizzazione.
A seguito di approfondite ricerche si è stabilito che la causa di queste diverse prestazioni è dovuto all'inserimento di ritardi da parte del codice JNI dell'applicazione Android.
Come chiarito anche da IBM in un approfondimento sulle *Java Native Interface* [40], il codice JNI viene interpretato dalla *Dalvik Virtual Machine*, macchina virtuale che si occupa dell'esecuzione del codice Android, inserendo di conseguenza dei ritardi nell'applicazione, indipendenti dal codice in questione.
Nel grafico in Figura 20 viene illustrato il rapporto tra il time slot e le medie sia della probabilità di bit inviati correttamente sia del numero di errori delle trasmissioni osservate finora.

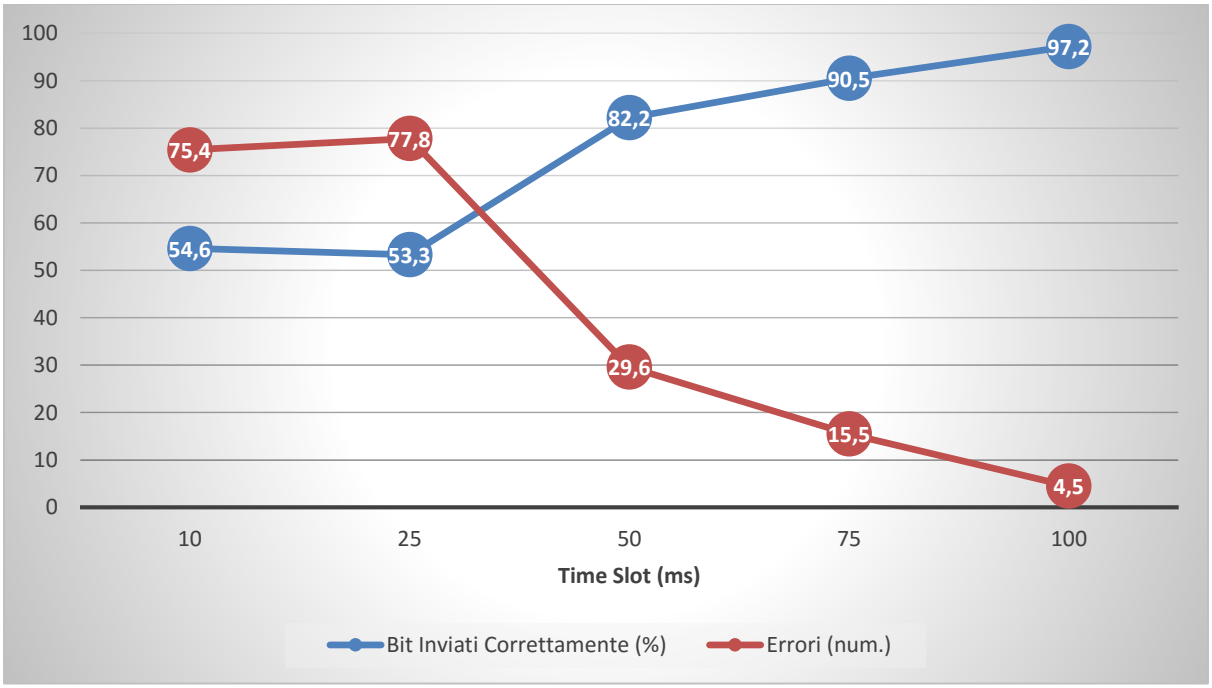

Figura 21 Grafico che mostra la relazione tra time slot e prestazioni del covert channel

Il grafico evidenzia, inoltre, il rapporto di proporzionalità diretta tra prestazioni e time slot; al crescere di quest'ultimo si ottengono risultati migliori e una migliore affidabilità della trasmissione, in grado di inviare correttamente, nella maggior parte dei casi, il messaggio celato.
Il secondo messaggio utilizzato per la sperimentazione è una sequenza numerica di 10 cifre, che potrebbe rappresentare un numero di cellulare, un codice bancario o un qualsiasi altro identificatore che potrebbe essere d'interesse ad un hacker.
Per la verifica è stata utilizzata la stringa "7155493201".
Vengono di seguito riportate le tabelle che mostrano i dati relativi ai test con i diversi time slot.

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	100	0	152
2	100	0	152
3	100	0	152
4	100	0	152
5	100	0	152
6	100	0	152
7	100	0	152
8	100	0	152
9	100	0	152
10	100	0	152

Tabella 29 Invio del codice numerico a 100 ms

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	66	19	152
2	100	0	152
3	100	0	152
4	100	0	152
5	100	0	152
6	85	8	152
7	100	0	152
8	100	0	152
9	100	0	152
10	100	0	152

Tabella 30 Invio del codice numerico a 75 ms

Risulta evidente già dai primi dati, elencati nelle tabelle 29 e 30, come la comunicazione presenti un tasso più elevato di affidabilità rispetto alla trasmissione del messaggio precedente.
Questo mutamento delle prestazioni è dovuto alla lunghezza minore del messaggio covert trasmesso, che implica un numero minore di caratteri da inviare e di conseguenza una minore probabilità di perdita di sincronismo tra le due parti in comunicazione.
Questa variazione emerge in maniera ancora più sorprendente se si considera la spedizione del messaggio con un time slot di 50 millisecondi, che riesce a perseguire l'obiettivo in tutti i tentativi senza incorrere in nessun errore.

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	100	0	152
2	100	0	152
3	100	0	152
4	100	0	152
5	100	0	152
6	100	0	152
7	100	0	152
8	100	0	152
9	100	0	152
10	100	0	152

Tabella 31 Invio del codice numerico a 50 ms

Questo dato permette di interpretare in una luce diversa i due tentativi che hanno presentato errori con l'*interpacket gap* a 75 millisecondi, causati molto probabilmente da una calo di velocità nella rete durante la trasmissione.
Riducendo ulteriormente il time slot si incorre, inevitabilmente, in un abbassamento delle prestazioni, minore rispetto al precedente esperimento ma comunque significativo, poiché non permette una codifica integrale e corretta del messaggio celato. Una minoranza dei caratteri, infatti, non viene decodificata in maniera corretta, facendo perdere di significato parte del messaggio celato.

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	66	27	152
2	72	22	152
3	95	4	152
4	67	26	152
5	66	27	152
6	95	4	152
7	85	12	152
8	51	39	152
9	48	41	152
10	67	26	152

Tabella 32 Invio della sequenza numerica a 25 ms

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	75	37	152
2	69	47	152
3	69	46	152
4	69	47	152
5	23	114	152
6	69	47	152
7	76	36	152
8	54	69	152
9	71	43	152
10	70	45	152

Tabella 33 Invio del messaggio a 10 ms

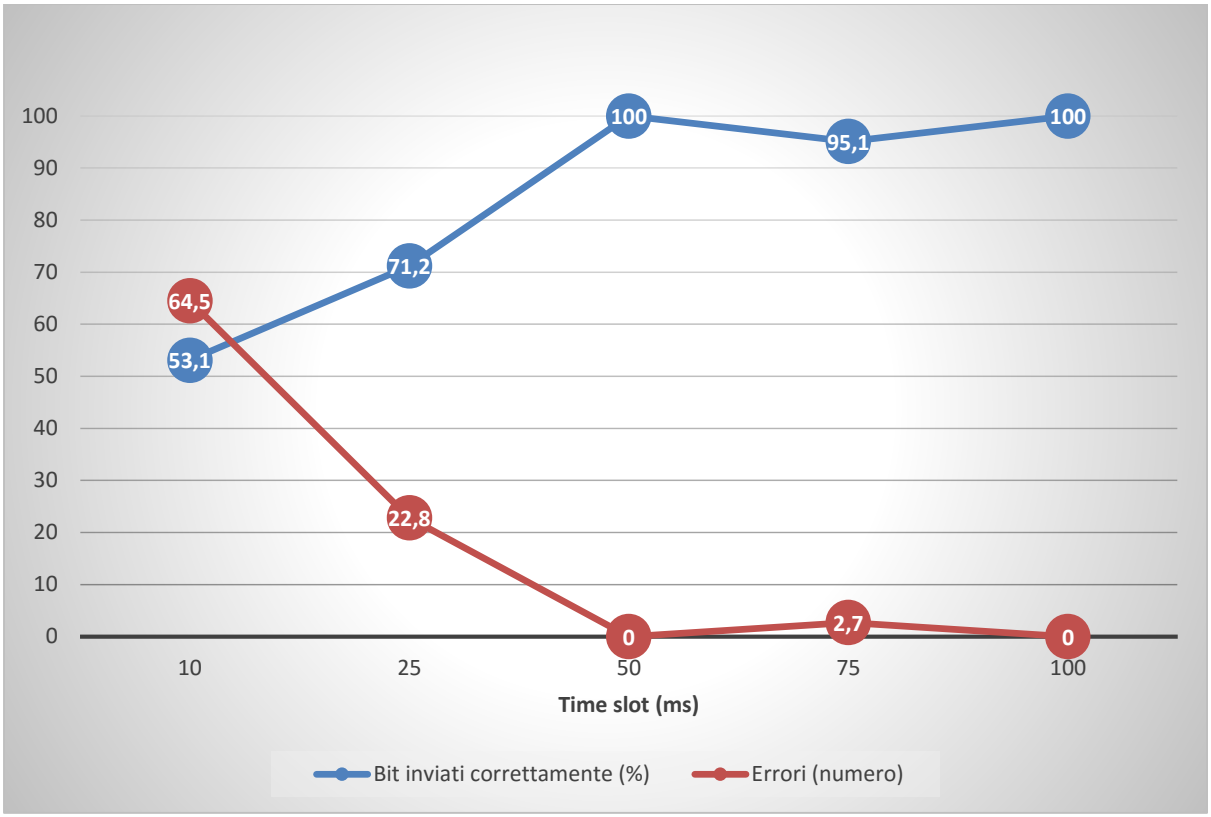

Figura 22 Il grafico mostra il rapporto tra le prestazioni e il time slot per la sequenza numerica 7155493201

Il grafico in Figura 21 riassume le prove sperimentali condotte sulla sequenza numerica, ribadendo lo stretto legame che intercorre tra *interpacket gap* e prestazioni del canale nascosto.
L'ultimo messaggio utilizzato per i test è "Codice segreto 1234", scelto non per la lunghezza o la difficoltà implicita del messaggio ma per la presenza di spazi che influiscono in maniera negativa sulla sincronizzazione.
Infatti, questo speciale carattere produce una lunga sequenza di zeri (in binario risulta "00100000") che mantengono in un lungo stato di pausa il client che potrebbe, di conseguenza inserire dei ritardi al momento del "risveglio", compromettendo la trasmissione.

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	98	2	296
2	99	1	296
3	100	0	296
4	99	1	296
5	100	0	296
6	100	0	296
7	100	0	296
8	99	1	296
9	100	0	296
10	99	1	296

Tabella 34 Invio del messaggio a 100 ms

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	99	1	296
2	96	9	296
3	100	0	296
4	99	1	296
5	100	0	296
6	100	0	296
7	97	7	296
8	99	1	296
9	100	0	296
10	99	1	296

Tabella 35 Trasmissione dell'informazione a 75 ms

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	99	1	296
2	100	0	296
3	96	9	296
4	100	0	296
5	99	2	296
6	97	8	296
7	100	0	296
8	99	1	296
9	100	0	296
10	100	0	296

Tabella 36 Trasmissione a 50 ms

I dati riportati nelle tabelle 34, 35 e 36 mostrano come un alto valore del time slot permetta alla trasmissione di non essere influenzata in maniera drastica da questo fenomeno, mantenendo un elevato livello di affidabilità.
Sono stati rilevati, però, un numero maggiore di errori rispetto al precedente test, che denotano come la sincronizzazione sia più difficile da mantenere in questo caso.
Al contrario, abbassando l'intervallo di tempo tra un datagramma e il successivo a 25 e 10 millisecondi, si riscontra un peggioramento notevole così come riportato nelle tabelle 34 e 35.

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	91	24	296
2	40	177	296
3	53	139	296
4	41	174	296
5	60	116	296
6	69	90	296
7	91	24	296
8	70	88	296
9	53	139	296
10	41	174	296

Tabella 37 Trasmissione del messaggio a 25 ms

Tentativo	Bit Inviati Correttamente (%)	Errori (num. Bit)	Bit Totali (numero)
1	52	142	296
2	38	181	296
3	40	175	296
4	64	104	296
5	51	144	296
6	52	142	296
7	50	158	296
8	45	160	296
9	64	104	296
10	51	144	296

Tabella 38 Trasmissione del messaggio a 10 ms

In questi tentativi, inoltre, si è manifestato l'errore di tipo *bit flip*, ovvero l'errata decodifica del valore di un bit che causava l'interpretazione del numero 5 piuttosto che del 4 nell'ultimo carattere della sequenza.

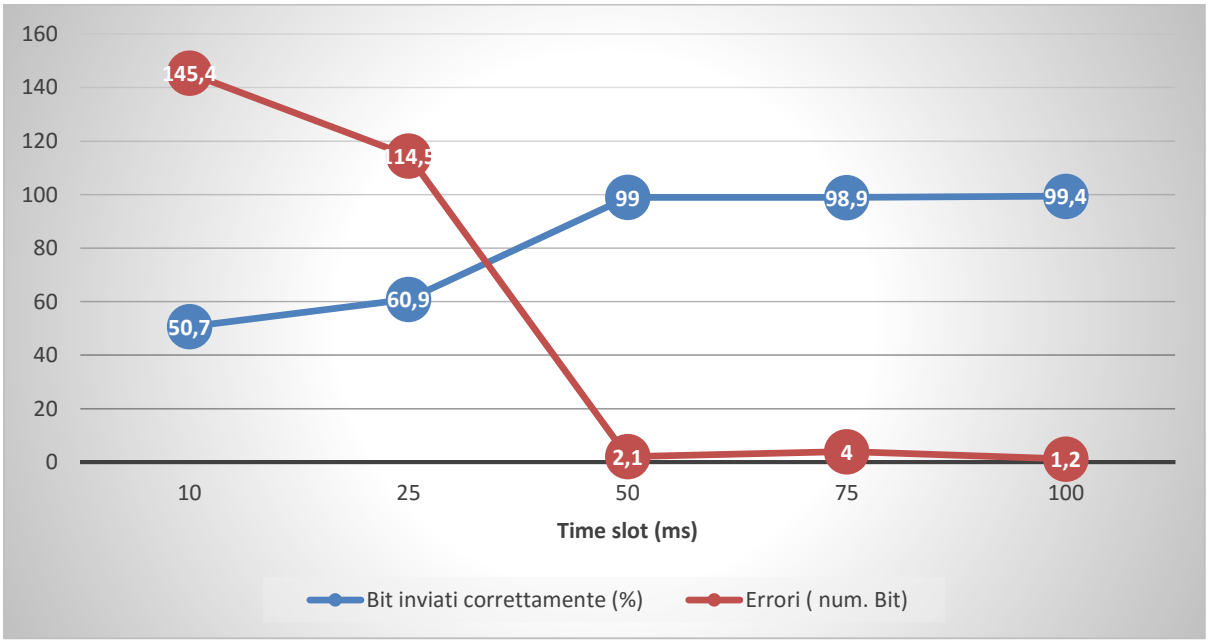

Figura 23 rapporto tra prestazioni ed errori per la trasmissione del messaggio con gli spazi

Il grafico, mostrato in Figura 23, come nei precedenti casi, riassume le prestazioni osservate durante la trasmissione della suddetta stringa, presentandone la percentuale di bit corretti inviata media che, come pronosticato, diminuisce all'abbassamento del valore del time slot.

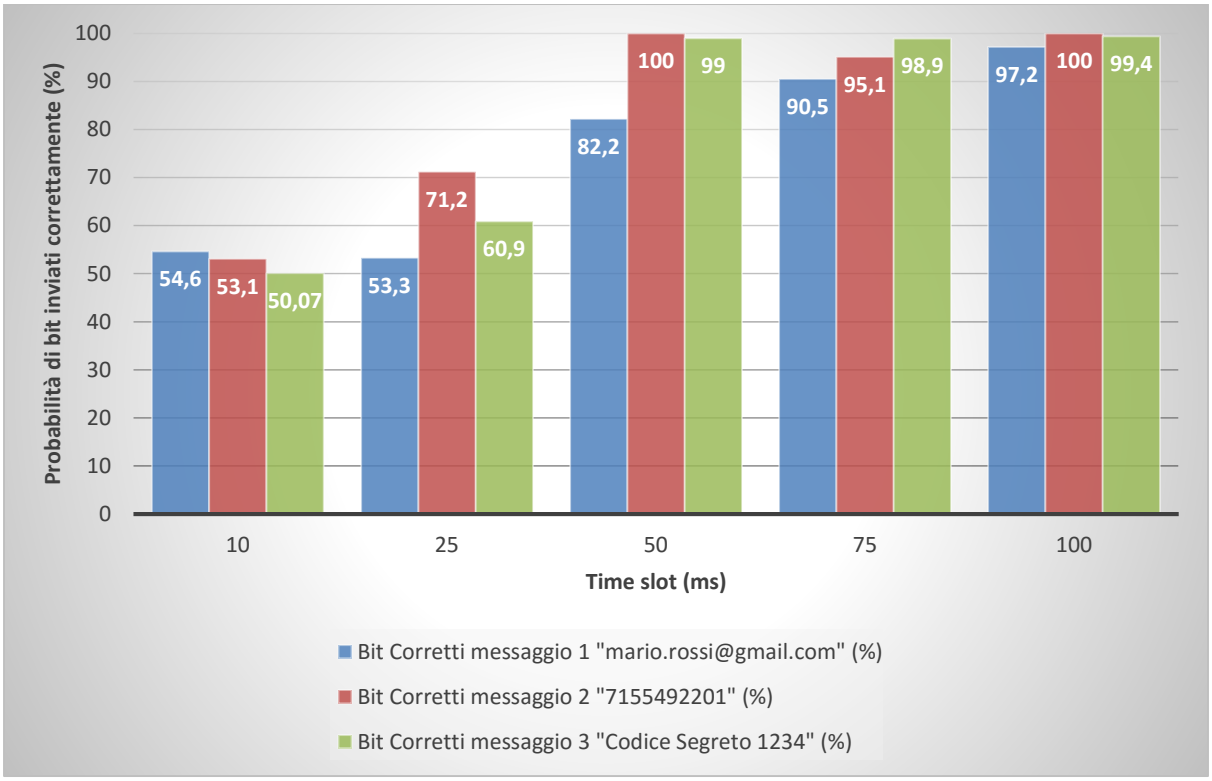

Figura 24 Prestazioni osservate durante l'invio di 3 differenti messaggi.

Il grafico in Figura 24 compara le prestazioni dell'invio dei 3 messaggi, riportando le percentuali dei bit corretti inviati, per ogni messaggio, al variare del time slot.
Si può osservare come, in media, le prestazioni più basse siano quelle rilevate durante la trasmissione del messaggio 3, che provoca la perdita più frequente di sincronizzazione tra sender e receiver, fenomeno dovuto alle lunghe sequenze di bit 0 introdotte dalla spaziatura dai caratteri.
Il grafico riporta anche i casi in cui si sono verificati degli aumenti di latenza nella rete, osservabili in un calo delle prestazioni rispetto alla trasmissione dello stesso messaggio con un time slot inferiore.
Per intervalli temporali ampi, invece, le prestazioni sono ottime qualsiasi sia il messaggio che si sta inviando, fattore che consente, così, la ricezione e la decodifica corretta di un messaggio illecito da parte di un server.

Conclusioni e Sviluppi Futuri

In questa tesi è stato introdotto il concetto di covert channel, di cui si sono analizzati i punti di forza e debolezza.
La realizzazione di un covert timing channel in ambiente Android ha, inoltre, permesso di esaminare l'effettiva portata ed efficienza di questo attacco.
L'applicazione implementata, infatti, ha evidenziato una buona affidabilità nella trasmissione di un messaggio illecito, in maniera totalmente nascosta, per intervalli tra un datagramma e il successivo che oscillano dai 50 ai 100 millisecondi.
Per intervalli temporali più bassi, al contrario, c'è stato un calo delle prestazioni che ha reso, nella maggior parte dei casi, inutilizzabile il contenuto del messaggio celato.
La sincronizzazione tra sender e receiver, come illustrato nel corso della tesi, si è confermata, quindi, la principale debolezza strutturale di un canale nascosto.
La riuscita della comunicazione, in un ampio numero di casi, evidenzia, però, come questa tipologia di attacchi sia una minaccia reale e che necessita, quindi, di uno studio più approfondito al fine di trovare delle contromisure adatte per rendere questa offensiva inefficace.
Le potenzialità di questo tipo di attacco sono molto alte, fattore influenzato anche dalla miriade di implementazioni possibili. I sistemi mobili, infatti, offrono una elevata varietà di risorse che possono essere utilizzate per attuare una comunicazione nascosta di questo tipo, come per esempio la messaggistica o le chiamate voce, funzionalità utilizzate quotidianamente da moltissimi utenti; elementi che potrebbero far aumentare la diffusione di questi attacchi in futuro.
L'applicazione sviluppata e presentata in questa tesi ha introdotto del traffico "lecito" per inviare un messaggio celato.
Un possibile sviluppo del covert channel implementato potrebbe essere, di conseguenza, l'utilizzo di traffico preesistente, fattore che renderebbe molto più complessa l'eventuale rilevazione del canale nascosto, elevandone il livello di pericolosità.
Inoltre, l'applicazione realizzata ha adoperato un pattern di sincronizzazione che inserisce un *overhead* consistente nella trasmissione; la ricerca, quindi, di un algoritmo di sincronizzazione più performante permetterebbe di abbassare, ulteriormente, l'intervallo di tempo tra un datagramma e il successivo, aumentando, di conseguenza, la banda del canale e la difficoltà nella *detection* di questa modalità di trasmissione, migliorandone l'affidabilità.
Un altro sviluppo interessante, che potrebbe essere analizzato, è l'instaurazione del canale nascosto tra due dispositivi mobili, realizzando, di conseguenza una applicazione che svolga le funzioni tipiche di un server, come la decodifica degli intervalli temporali tra i pacchetti al fine di decifrare il messaggio celato.
La realizzazione di questa nuova modalità di covert channel porterebbe alla luce tutta una serie di nuove problematiche e considerazioni, aumentando a dismisura la portata dei canali nascosti e introducendo la possibilità di compromettere le informazioni sensibili disponibili su un altissimo numero di dispositivi, data l'enorme popolarità e diffusione degli smartphone e dei tablet.

Riferimenti

[1] CCM. *Introduzione alla sicurezza informatica.*
Available on:
http://it.ccm.net/contents/833-introduzione-alla-sicurezza-informatica
[2] InfoSecurity Ireland.
Available on:
http://www.isin.ie/go/clusters/security/infosecurity-ireland-isi-
[3] Symantec, *Threat Report.*
Available on:
http: www.symantec.com/security_response/publications/threatreport.jsp
[4] Verizon, *Data Breach Investigation Report 2015.*
Available on:
http://www.verizonenterprise.com/DBIR/2015/
[5] Trucioli Savonesi. *Virus del pc? Chiamateli malware.*
Available on:
http://www.truciolisavonesi.it/articoli/numero153/ferraris.htm
[6] Kaspersky Lab. *A malware classification 2013.*
Available on:
https://blog.kaspersky.com/a-malware-classification/3037/
[7] Nsfocus Blog. *DDoS Attack and Defense 2012.*
Available on:
http://nsfocusblog.com/2012/10/29/ddos-attack-and-defense/
[8] Haking and Security. *All about DNS poisoning.*
Available on:
http://hakingandsecurity.blogspot.it/2013/11/all-about-dns-poisoning.html
[9] Chris Baley. TRENDMICRO Blog. *Extended validation certificates warning against mitm attacks.*
Available on:
http://blog.trendmicro.com/trendlabs-security-intelligence/extended-validation-certificates-warning-against-mitm-attacks/
[10] Wikipedia. *Sicurezza Informatica.*
Available on:
https://it.wikipedia.org/wiki/Sicurezza_informatica
[11] CryptoSmith. *Introduction to MLS.*
Available on:
http://cryptosmith.com/mls/intro/
[12] Centos. *Deployement guide.*
Available on:
https://www.centos.org/docs/5/html/Deployment_Guide-en-US/sec-mls-ov.html
[13] University of Cambridge. *Multilevel Security.*
Available on:
https://www.cl.cam.ac.uk/~rja14/Papers/SEv2-c08.pdf
[14] Quark Security.
Available on:
https://quarksecurity.com/
[15] Dimensional Research, *Mobile security survey 2013.*
Available on:
http: www.checkpoint.com/downloads/products/check-point-mobile-security-survey-report2013.pdf

[16] Dan Godin. Ars Technica. *New Android lockscreen hack gives attackers full access to locked devices 2015.*
Available on:
http://arstechnica.com/security/2015/09/new-android-lockscreen-hack-gives-attackers-full-access-to-locked-devices/
[17] Information Week. *Evolution of mobile malware*.
Available on:
http://www.informationweek.com/mobile/mobile-malware-protect-yourself-against-evolving-threats/d/d-id/1099438
[18] Google.inc. *Security Tips.*
Available on:
http://developer.android.com/training/articles/security-tips.html
[19] Apple.inc. *Listed of available trusted root certificates in iOS.*
Available on:
https://support.apple.com/en-us/HT204132
[20] Lampson, B.W., *A Note on the Confinement Problem.*Communications of the ACM, Oct.1973.16(10): p. 613-615.
Available on:
http://research.microsoft.com/en-us/um/people/blampson/11-Confinement/Acrobat.pdf
[21] Simmons, G.J.: *The Prisoners Problem and the Subliminal Channel. In: Proceedings Advances in Cryptology*, pp. 51–67 (1983).
[22] Slideshare. *Image Stenography using mode 16 method.*
Available on:
http://www.cs.jhu.edu/~fabian/courses/CS600.624/slides/CovertTimingChannel.pdf
[23] John C. Wray,*An analysis of covert channels.*
Available on:
http://www.cs.cornell.edu/people/vickyw/iflow/papers/wra91.pdf
[24] The University of West Indies. *Introduction to TCP/IP*.
Available on:
http://www.eng.uwi.tt/depts/elec/staff/kimal/tcpip.html
[25] Craig G. Rowland, *Covert Channels in TCP/IP protocol suite.*
Available on:
http://firstmonday.org/ojs/index.php/fm/article/view/528/449
[26] Wade Gasior, Li Yang. *Exploring Covert Channel in Android Platform.*
[27] Cabuk, S., Brodley, C.E., Shields, C.: *IP Covert Timing Channels: Design and Detection. In: Proceedings of 11th ACM Conference on Computer and Communications Security* (CCS), pp. 178–187, (2004)
[28] Padlipsky, M.A., Snow, D.V., Karger, P.A.*: Limitations of End-to-End Encryption in Secure Computer Networks. Technical report,* ESD-TR-78-158 (1978)
[29] Esser, H., Freiling. F.: *Kapazitaetsmessung eines verdeckten Zeitkanals ueber HTTP. Technical report*, TR-2005-10 (2005)
[30] Murdoch, S.J.: *Hot or Not: Revealing Hidden Services by Their Clock Skew. In: Proceedings of 13th ACM Conference on Computer and Communications Security (CCS)*, pp. 27–36. IEEE Press, New York (2006)
[31] Corporacy, I. D. (2015). *Smartphone OS Market Share, 2015 Q2.*
Available on:

http://www.idc.com/prodserv/smartphone-os-market-share.jsp
[32] Wikipedia. "Sistema Client/Server".
Available on:
https://it.wikipedia.org/wiki/Sistema_client/server
[33] Università degli Studi della Tuscia. *Principi base della rete Internet.*
Available on:
http://www.unitus.it/info/help/internet2000/testo/29_testo.htm
[34] Wikipedia. (s.d.). *Android Studio.*
Available on:
https://en.wikipedia.org/wiki/Android_Studio
[35] Google.inc.
Available on:
https://developer.android.com/ndk/guides/android_mk.html)
[36] Tenouk.com. *Network Programming: Linux Socket part 11*.
Available on:
http://www.tenouk.com/Module41a.html
[37] Google.inc. *App Manifest.*
Available on:
http://developer.android.com/guide/topics/manifest/manifest-intro.html
[38] Google.inc. (s.d.). *Official Android Guide.*
Available on:
http://developer.android.com/reference/android/os/AsyncTask.html
[39] Ookla. Test available on:
http://www.speedtest.net/it/
[40] IBM. *Best practices for using the Java Native Interface.*
Available on:
http://www.ibm.com/developerworks/library/j-jni/